2016

Strategies for including cloud-computing into an engineering modeling workflow

Sunil Suram
Iowa State University

Strategies for including cloud-computing into an engineering modeling workflow

by

Sunil Suram

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Program of Study Committee:
Kenneth M. Bryden, Major Professor
Arne Hallam
Richard A. Lesar
Mark Mba-Wright
Abhijit Chandra

Iowa State University

Ames, Iowa

2016

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Mark Bryden for his guidance and patience during my graduate studies. Without his vision, mentorship and, help, it would not have been possible to conceive of and implement several of the ideas in this dissertation. I would also like to thank my committee members Dr. Richard LeSar, Dr. Abhijit Chandra, Dr. Mark Mba-Wright and Dr. Arne Hallam for their input.

Needless to say, the completion of this dissertation would not have been possible without the infinite patience and support of my wife Mukta. To my daughter Mishika, who wondered where *papa* was during the last few months, lots of love and looking forward to many new beginnings with her. To my parents Umamaheswaram and Jayabhagya Suram and my sister Ragini, for their unconditional love and support over the years.

Many thanks to Saurav for his friendship. I would also like to thank my family and friends for their love and support.

Several quotes have kept me going over the years and this one by Mark Twain is one of my favorites', *"Never let your schooling interfere with your education."*

CHAPTER 1. GENERAL INTRODUCTION

1.1. Introduction

With the advent of cloud computing, high-end hardware is now universally accessible on an on-demand basis. Consumer and enterprise applications are being re-architected to utilize this new way of computing. The fields of engineering modeling and computational science have utilized cloud computing primarily as a pool of computing, storage, and networking resources that can be added and removed as needed (Vöckler et al. 2011, K. Jorissen et al. 2012). Most of the research within engineering and scientific computing has thus focused on using the cloud as a less expensive alternative to purchasing hardware and comparing its performance with traditional compute clusters. However, there are other cloud computing opportunities for building novel engineering and scientific analysis, visualization, and data management applications.

With the on-demand availability of cloud computing resources, it has become easier to build and run detailed computational models to solve engineering problems (Iosup et al., 2011, Jorissen et al. 2012). As a result, massive amounts of data is created from computational fluid dynamics (CFD), finite element analysis, and other computational techniques (Liu et al. 2015). However, these models have high computational costs and are of limited use for engineering analyses that require a relatively quick turnaround for detailed engineering as well as a rapid turnaround for performing "what-if" analysis during conceptual and preliminary design. Thus today the demands for an iterative engineering workflow are largely unmet despite advances in computing capabilities. Additionally, there is a need for non-traditional data sources and models to be seamlessly coupled with engineering models using cloud computing platforms.

In this dissertation, engineering modeling is considered from the perspective of Internet-based applications running on a cloud infrastructure as opposed to a monolithic software or code

written to solve a specific engineering problem. From this vantage point, this dissertation seeks to reframe the question “how are detailed models used in engineering design and decision making?” to the question “how should engineering workflows be able to utilize models given the present day cloud platforms?” When asked this way, it is clear that detailed models need to be deployable in such a way that they can be used directly in the design and decision-making process. Additionally, this cloud-based engineering workflow must reduce the amount of work and complexity for the producers of the models, the system builders who today use the computational results (often as reduced order models), and the consumers of the systems models. Thus, engineering decision-making entails the following requirements:

1. The models and data developed for an engineering application should be deployed as an Internet based service.
2. The models used in the design process should be easily composable into complex systems of models capable of answering critical engineering questions. That is, models should be able to be invoked as and when needed.
3. These models must be readily publishable by their developers for use in systems models and analyses.
4. Models must be able to exchange information with each other via an intermediary service and must be able to join a federation of models in order to solve a larger set of problems.
5. To reduce the time needed to compute an answer from a detailed model, hybrid models consisting of detailed computational models and reduced order models (ROMs). ROMs should be constructed on-the-fly in a manner that is transparent to the user and compatible with systems modeling.

6. Engineering teams should be able to substitute one model in a system of models for another model (with more or less detail or information) without disrupting the system of models.

Implementing these principles requires a fundamental change in the current modeling and design paradigm. In today's modeling and design paradigm

- Most detailed models enter the design and decision making process as a single piece of information (e.g., the maximum stress on a component, the maximum temperature, etc.), a conclusion (e.g., the optimal thickness), qualitative guidance to be used in conjunction with engineering judgment; or after a set of detailed reviews, a reduced order model in larger systems model.
- When a detailed model is needed the starting point is often an entirely new model.
- The creation of a large, complex systems model requires the development of a global ontology, a set of coupling protocols that is accepted and used by model builders at their points of interaction between the models.

This dissertation proposes a novel engineering workflow that addresses the first two aspects of the current modeling and design paradigm by

1. Developing a framework using which engineers can use stateless computational models as services running on the Internet. Utilizing this framework, multiple constituent models can be linked together to create more complex, composite engineering models.
2. Creating hybrid models (information artifacts) composed of detailed models and reduced order models (ROMs) that are transparent to the user and system analysts.

The ability to choose between multiple compatible models and substitute one for the other on-the-fly, although clearly possible from the framework developed here, is not

demonstrated. In addition, the question of ontological and semantic independence is not addressed in this dissertation but is left for future research efforts.

1.2. Dissertation Organization

This dissertation consists of five chapters with chapters 2, 3 and 4 formatted as journal articles focusing on the methodology and results from this research work. Chapter 2 has been published in the journal Advances in Engineering Software, and chapters 3 and 4 will be submitted for peer review in the same journal.

Chapter 2 discusses a methodology developed to utilize data generated from high-fidelity models to construct ROMs and incorporate them into an iterative engineering design workflow. As a part of this, the article introduces the concept of a ROM server and describes in detail its ability to enable seamless communication between the consumers of detailed analysis (the engineering designers and decision makers) and the producers of detailed analysis (the analysts). It also clarifies the roles of producers of the models and the consumers of the results of the models and uses the ROM server as the point of interaction between them. Several cases with varying number of producers and consumers of engineering data are considered, and an analysis is presented that demonstrates the efficiency of the ROM server. This framework is then demonstrated using the engineering design of the shape of heat exchanger fins.

Chapter 3 introduces the concept of federations of web-enabled models that can be assembled and managed via the federation management system (FMS). Within this architecture each of the constituent models is an independent web-based model service accessible via a web API using integration protocols chosen by the model developer. The developed modeling architecture is a decentralized system at its core where constituent models are treated as independent functional units that solve a particular problem. The constituent models are also

required to publish their input and output data formats. This is in essence a micro-service architecture. In addition, the models are stateless i.e., they do not persist state beyond the duration of the computation that is being performed, which allows their easy reuse by the same system model or another system model. Thus the user has the ability to choose individual models and link them with one another, and the FMS ensures that they are invoked and executed in the intended sequence. This approach is a departure from the traditional "library" approach where a monolithic piece of code is used to integrate several software libraries to solve a specific engineering problem. Finally, this federated model framework is demonstrated by solving the problem of linking individual models in the design of small wood-burning cookstoves.

In chapter 4, the concepts introduced in chapters 2 and 3 are combined to create a hybrid modeling element (information artefact) that includes both a detailed model and a ROM of the detailed model. This hybrid model can be utilized as a web-based model service and is available to be included as a single information artefact in a federation of models. That is, the user and/or the system simply calls the model and receives the result in the same way as any other web-enabled model within the federation. The creation and maintenance of the ROM and the choice of using the detailed model or the ROM are handled by the hybrid model with constraints provided by the FMS. This logic is introduced into the FMS as a set of substitutability rules, which are defined by a system builder and given to the FMS. This framework is demonstrated using the heat exchanger design problem used in chapter 2.

Finally, chapter 5 summarizes the results and then discusses the potential of this research. It also points to future areas of research and a broad set of opportunities where further investigation is needed.

REFERENCES

J. Vöckler, G Juve, E Deelman, M Rynge and G. B. Berriman, 2011. Experiences Using Cloud Computing for a Scientific Workflow Application, Proceedings of the 2nd international workshop on scientific cloud computing, pp 15-24, ACM New York, NY, USA.

K. Jorissen, F.D. Vila and J.J. Rehr, 2012. A high performance scientific cloud computing environment for materials simulations, Computer Physics Communications, vol 183, issue 9, pp 1911-1919.

A. Iosup, S. Ostermann, et al., 2011. Performance Analysis of cloud computing services for many-tasks scientific computing, IEEE Transactions on Parallel and Distributed Sytems, vol. 22, no. 6, pp. 931-945.

J. Liu , E. Pacitti, P. Valduriez, M. Mattoso, 2015. A Survey of Data-Intensive Scientific Workflow Management, Journal of Grid Computing, Dec. 2015, vol. 13, issue 4, pp 457-493.

CHAPTER 2. INTEGRATING A REDUCED-ORDER MODEL SERVER INTO THE ENGINEERING DESIGN PROCESS

Modified from a paper published in *Advances in Engineering Software*, *(90), 2015*

Sunil Suram and Kenneth M. Bryden*

Abstract

Engineering design is a complex and iterative process that involves multiple engineering teams sharing and communicating information during the design process. One aspect of engineering design involves the development of physics-based models and their analysis via numerical simulations that are computationally expensive. To overcome the time constraints due to the complexity of numerical simulations, reduced-order models (ROM) such as proper orthogonal decomposition are being increasingly used. Decreasing the simulation time, however, does not address the inefficiencies in communicating engineering models and analysis during the design process. This paper proposes developing and incorporating a ROM server into the design workflow. The ROM server stores all data associated with a given engineering model and automatically constructs a ROM every time a model is created or updated, thus maintaining a consistent version of information across multiple engineering teams. A common engineering workflow is compared with one using a ROM server. A cost of synchronization metric has been defined based on the parameters of data size, size of the engineering team and design iterations. This metric has been evaluated and compared for the cases with and without a ROM server and it was found that the cost of synchronization is lower when a ROM server is used in the design workflow. It is shown that as the team size increases, the ROM server helps with more efficient

information storage and transfer. Finally, an example problem of a heat-exchanger fin shape design is used to demonstrate the ROM server framework.

2.1. Introduction

Engineering is increasingly dependent on modeling and simulation for design, optimization, and many other engineering decision-making tasks. With increasing computational capabilities it has become easier to run large-scale, physics-based high-fidelity simulations and analyze various physical phenomena. Examples of these simulation techniques include computational fluid dynamics (CFD), finite element analysis, and molecular dynamics. Often this involves discretizing the domain into a finite number of grid points and solving the governing, coupled partial differential equations over the discretized domain. Resolving physical phenomena at these levels of fidelity is time-consuming, and the computational complexity is a function of the number of grid points considered, the complexity of the geometry, and the physics represented by the equation set. As a result, running the numerous "what-if" scenarios needed to support a simulation-based engineering and design process is often computationally prohibitive.

To overcome this limitation, various data-driven reduced-order modeling techniques have been developed. Data-driven reduced-order modeling relies on first creating a collection of computational solutions to construct a set of basis functions. These basis functions are then used to make evaluations of the reduced-order model (ROM) in lieu of using the original large-scale physics-based computational model (Samadiani et al., 2010; Everson et al. 1995; My-Ha et al., 2007; Sakurai et al., 2006). One reduced-order modeling technique is proper orthogonal decomposition (POD). POD-based ROMs have been used in a number of engineering and scientific applications, including fluid mechanics (Suram et al., 2008; Tan et al., 2003; My-Ha et

al., 2007; Astrid, 2004) and solid mechanics (Zhou and Hitt, 2011). For example, Suram et al. constructed a ROM based on CFD simulations of flow through a mixing nozzle (Suram, McCorkle and Bryden, 2008). Samadiani et al. reviewed design methodologies for data-center server thermal management based on the POD method (Samadiani and Joshi, 2010). Willcox et al. developed an inverse design technique based on a POD technique for incomplete data (Bui-Thanh, Damodaran and Willcox, 2004). Zhou et al. used POD to analyze turbulent flow structures in a reacting jet (Zhou and Hitt, 2011). Du et al. have incorporated the POD technique into a finite difference scheme and have analyzed the errors after applying this unified scheme to develop a ROM for a chemical vapor deposition reactor (Du et al., 2011). The generality of the POD methodology also makes it useful in any field that involves studying and analyzing patterns of data that have been collected either computationally or experimentally. One interesting example where POD-based ROMs have been used with experimental data is in Chen et al. where the authors have analyzed bat flight kinematics from data collected from video samples (Chen, Kostandov et al., 2009). In engineering design, the focus of POD is to reduce the time to run the computational models within the engineering workflow. That is, the high fidelity models are created and are then used to create the ROM. The ROM is then used in the design process. Within this process it is often assumed that the ROM creation process is a linear and static process and that design exploration, optimization, and decision-making wait for the creation of the ROM. And that once the ROM is created, it does not change. However, engineering design is a dynamic process of exploration and consideration of multiple design options within a broad analysis space. Because of this, detailed high fidelity computational modeling is often delayed until the design is nearing completion. Pushing this detailed modeling forward in the design process has the potential to reduce costs and improve designs, but it requires a framework in

which the existing computational results can be utilized and updated while the analysis process proceeds. That is, the analysts and engineers need a common framework that enables a shared design-analysis workflow.

To meet this need, this paper proposes a client-server based architecture to build and evaluate POD-based ROMs in which the inputs from multiple engineers and analysts are incorporated and managed as a part of a dynamic and shifting engineering design process. This ROM server enables independent insights obtained by the designers and analysts within each iteration of a design to be used to improve successive iterations. In addition, integrating the various steps into a cohesive workflow enables faster, more consistent, and more predictable information sharing within the engineering design team, which may result in shorter more effective design cycles.

2.2. Background

Engineering design is an iterative decision making process in which collaborative groups of designers and engineers work together from the conceptual design to a final product. Many engineering design workflows have been proposed but most of these are similar to Figure 2.1 (Pahl et al., 2007; Ertas et al., 1996). As shown in Figure 2.1, the design process is composed of three main stages; (1) problem definition, (2) engineering design, and (3) design validation and verification. The engineering design stage can be thought of as consisting of three phases: conceptual design, preliminary design, and detailed design (Pahl et al., 2007; Ertas et al., 1996). During conceptual design engineers explore the design space through the generation of concepts that then are filtered using the constraints defined for the problem. Following conceptual design, preliminary design further refines these concepts to one design. During the detailed design phase the chosen design is optimized and finalized. High fidelity modeling offers the power to improve

and support creative engineering design in the exploration of ideas, which occurs during the conceptual design and preliminary design phases. But because of the time and expense required to develop, execute, and process these high fidelity models, they are typically used primarily during the detailed design phase. In contrast to the task oriented approach to the development of high fidelity models, engineering

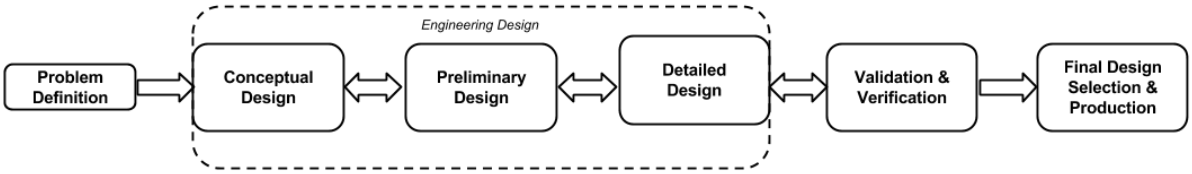

Figure 2.1. Workflow in an engineering design.

design is an iterative process in which the designers typically move back-and-forth between the conceptual, preliminary, and detailed design phases with relatively quick consideration and analysis of multiple designs, repeated iterations and expansions of proposed solutions, revisiting assumptions and decisions, and a series of design changes. Once completed a reduced set of designs are chosen for further verification and validation using additional analysis and field data. This can lead to changes to the initial design, thus requiring a repeat of the design stage. The exploratory and iterative nature of engineering design makes the process of communicating engineering information and analysis during the design stage between various engineering teams challenging.

2.2.1. Proper Orthogonal Decomposition

This section provides a brief discussion of the POD technique, which is needed to understand the implementation of the ROM server. Readers desiring a more detailed discussion of the POD technique are referred to (Kirby, 2001). The POD technique is used to find a set of optimal truncated orthogonal basis functions from a set of snapshot solutions. These snapshot solutions are solutions that span the space of interest. In engineering design these are typically from numerical simulations of the system or phenomena of interest. Within the space defined by the set of snapshot solutions, a solution vector, $\vec{x}_{sol}$, can be found using a set of basis functions, u^i.

$$\vec{x}_{sol} = \sum_{i=1}^{D} a_i u^i \tag{2.1}$$

where *D* is the dimension of the truncated vector space and the a_i are the coefficients that are used to compute the POD approximation for a given set of basis functions.

To find the optimal set of truncated basis functions needed for the ROM, the first step is to generate a dataset of *M* snapshot solutions that span the engineering design space of interest. The snapshot solution dataset is then centered by computing and subtracting the mean of the dataset from each snapshot. The mean-subtracted *M* snapshots are then concatenated in a single ensemble matrix, $\mathbf{X}_{N\ M}$, where *N* is the size of each snapshot and *M* is the number of snapshots. Once the ensemble matrix is assembled, the POD basis functions are computed from the covariance of the ensemble matrix using singular value decomposition. The coefficients are then found by projecting the POD basis functions onto the original ensemble matrix.

For a given ensemble matrix the basis functions are constant, and the coefficients are associated with the design space that was explored by the dataset of snapshot solutions used to

create the ensemble matrix. To evaluate a design for a set of parameters that are not a part of the dataset of snapshot solutions but are within the initial design space covered by the analysis, linear interpolation is performed on the coefficients as shown in Eq. (2.2). For example, if the coefficient vector, $\vec{a}^*$ at a given design parameter vector, $\vec{q}^*$ such that $\vec{q}_k < \vec{q}^* < \vec{q}_{k+1}$ have to be evaluated, $\vec{a}^*$ is given by Eq. (2.2). The POD approximation is then computed using the interpolated coefficients as described in Eq. (2.1).

$$\vec{a}^* = \vec{a}\big|_{q_k} + \left(\vec{a}\big|_{q_{k+1}} - \vec{a}\big|_{q_k}\right)\frac{\left(\vec{q}^* - \vec{q}_k\right)}{\left(\vec{q}_{k+1} - \vec{q}_k\right)} \tag{2.2}$$

The accuracy of the ROM is dependent on the number of terms used in the POD expansion and is determined by Eq. (2.3). E_i is the "energy" of the POD expansion and s_i represents the i^{th} singular value. It is called the energy because the singular values are equal to the square of the corresponding eigenvalues of the covariance matrix (Kirby, 2001).

$$E_i = \frac{s_i}{\sum s_i} \tag{2.3}$$

Thus the singular values of the covariance matrix $\mathbf{X}_{\text{cov}}$ can be used to determine if sufficient data is included in the initial snapshot set and to determine the accuracy of the ROM. For a design engineer evaluating a ROM, the singular value spectrum can serve as a useful guide to determine if there is sufficient information to make an engineering decision before proceeding to the next step in the design process. If the total energy captured by the dominant singular values is within the acceptable range of error, it can be concluded that the ensemble matrix has captured sufficient information. If this is not the case the ensemble matrix needs to be expanded with more information. The error of the ROM is a function of the number of models chosen to

construct the ensemble matrix that is dependent on the energy captured by the singular value spectrum (Kirby, 2001).

The POD technique is useful because it captures all the required information about the phase space of a given physical problem. When using it to solve an engineering design problem, this information can then be used in conjunction with the coefficient interpolation technique to explore the design space in a computationally efficient manner. This process can be summarized as follows:

- Identify the design parameters and design space of interest
- Create the computational data needed for the snapshot dataset that spans the design space of interest
- Create the POD coefficient and basis functions
- Make the POD ROM available for use
- Use the POD ROM to compute new solutions as needed to support the engineering design process

If the design space to be explored needs to be expanded or new aspects of the problem need to be explored, the snapshot dataset will need to be expanded and a new POD ROM will need to be developed. Additionally, the accuracy of the POD ROM increases as the number of snapshot solutions increases. Thus as the design process evolves and more accurate solutions are needed, the POD ROM will likely need to be updated in regions of the design space of particular interest.

The iterative nature of the design process and the continuing update of the ROM creates a communication challenge within the design and analysis team. To evaluate a ROM, the most recent set of coefficients and the basis functions need to be known by the user. If a user is

geographically in a different region or a part of a different engineering team interested in evaluating the ROMs or analyzing the results, this information has to be made available to them. Updating the ROM manually via email or download for local compute makes it challenging to ensure that the most recent ROM is used and that disparate members of the design group are using the same ROM. Furthermore, providing local access to this data for multiple users may not be possible. It is also likely that multiple POD ROM models would be used to address a large-scale complicated design problem, and a process is needed to coordinate the development and use of these multiple POD ROM models. This creates problems with management of data and version control. The ROM may remain on a single computational machine or may be exported to remote machines for simultaneous use. If it is kept on a single machine, access is limited because only one ROM computation can be performed at a time. If it is exported, maintaining version control of the ROM becomes difficult and different groups having conflicting or out-of-date information can slow the design process. In the next few sections we propose an engineering workflow to overcome these challenges and enable the seamless utilization of ROMs within the engineering design process.

2.2.2. Reduced-order modeling within traditional engineering design workflows

In a typical design workflow the design engineers identify the parameters of interest within the proposed design space. The analysts then computationally model the physical phenomena, verify and validate the models, and run the computer simulations needed. If ROMs are being used, the set of computer simulations is used to create the ROMs. The ROMs are then used to explore design space and optimize the design. Overall the creation of the model and the ROM is linear and unidirectional. This contrasts with the dynamic, exploratory nature of the

design process. The issues posed by working in a traditional batch compute paradigm to create the ROM include

- To work independently, the consumers of information have to make their own copies of the data. This imposes an inherent bottleneck in the collaborative process.
- If a producer of information makes changes to a computational model or design parameter, the consumers must explicitly synchronize their copies of the data to get the updated information.
- A part of this manual synchronization also involves the manual regeneration of the ROM.
- The creation of the snapshot solutions and the design proceed in series rather than in parallel, and hence analysis process and the design engineering process cannot inform each other of critical decisions and information in real time. Rather, the design waits for the analysis and the analysis waits for the design.

Thus, each of the activities involved are decoupled from one another, resulting in a discontinuous workflow and complicating the task of managing and effectively utilizing computational data.

One solution to the coordination and communication problem posed by the incorporation of high fidelity analysis in the design process is to develop a client-server based engineering design workflow. A client-server architecture has one central computer node acting in the role of a server. Clients connect to this server to request information or use the server's hardware resources to perform a computational task. Once this task is performed information is returned back to the client, completing the transaction. This is a centralized architecture, which implies that the global state needs to be maintained only on one node. This makes the maintenance of a single server relatively straightforward. Because the client typically does not perform the heavy

computations, this architecture has the advantage that small form-factor clients (e.g., tablets and other mobile devices) can be developed to perform specific design engineering tasks. Based on this, this paper presents a client-server based architecture to support the integration of high-fidelity modeling using POD ROM modeling into the engineering design workflow.

2.3. Proposed Engineering Workflow

As noted earlier, the engineering design process is a dynamic, information-rich process that brings together many disciplines to create a product or solution that addresses a given set of needs within a complex and constrained design space. Within this space computational models and information are created and used by varying groups at varying times. Within this simulation-based engineering design environment a workflow that simplifies the creation, use, and update of the ROMs and enables various engineering groups to share this information easily is needed. Figure 2.2 shows the proposed engineering design workflow in which a ROM is included as a natural part of the design process. As shown, the ROM server is central to all the interactions at various stages of the engineering design process. The computational models created and updated and the solutions explored during the design process are all stored in the ROM server. The solution sets generated are utilized to build the reduced-order models that the ROM server publishes for use by various teams. During the design and analysis stages, changes to design parameters, computational constraints, changes to geometry, and other changes can all be stored in the ROM server, which periodically reconciles this data to generate an updated lower-dimensional approximation. Various engineering groups can then use this database of computational models during the different stages of the design process to perform engineering tasks. For example, a CFD analyst can analyze a flow field and a design engineer can work on optimizing shape while yet another engineer can extend the design space by adding more

computational models to the database, all doing so simultaneously. In the same way, a design engineer can request that the ROM be extended or improved in a specific way. These interactions are marshalled by the ROM server in such a way that the producers and consumers of computational data have a consistent view of the current state and the requests for new or improved information, and they have access to the same data.

Several researchers have referred to the collection of computational results that are utilized to construct the POD basis functions as a “database” (Suram et al., 2008; Tan et al., 2003; My-Ha et al., 2007; Astrid, 2004; Gunzburger, 2002; Kerschen et al., 2005). Although in the strict sense of the word this collection of computational results is a database, from the perspective of database systems it is missing several key components.

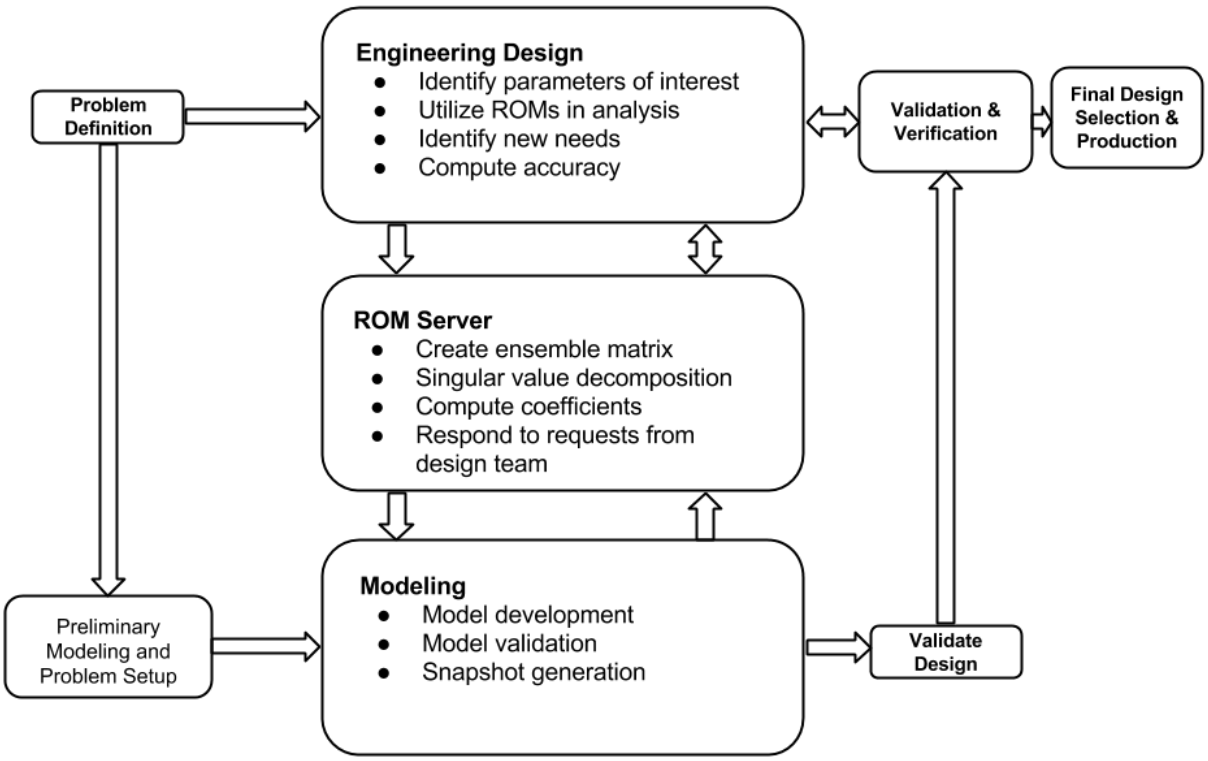

Figure 2.2. Workflow developed utilizing the ROM server.

It is not accessible to multiple users simultaneously and requires local access to all the data to perform evaluations. In addition, a database system addresses the issues of accessing the database, performing computations, and running optimization problems remotely across a computer network. In the cases cited here, the POD-based ROM evaluation process is a batch process (i.e., the computer program reads the necessary input data, computes the POD approximation for the given input, writes the output to a file and then exits). When the POD approximation needs to be computed for a different set of design parameters, the computer program is restarted. The same also applies to cases where computational datasets are added to or deleted from the ROM database. In these cases the entire database has to be recreated to account

for these changes and the POD basis functions have to be recomputed. Each user must then manually update their work, and as the number of users increases the cost of synchronization also increases. In a collaborative team environment this manual synchronization of data and information is a slow and inefficient process. Furthermore, because the run-time of a ROM is much quicker than that of a high-fidelity computational model, running ROMs as a batch process does not allow for efficient sharing of computational information and the fast computations associated with them.

Because a data-driven ROM relies on a computational database, it is an excellent candidate to be treated as a database of engineering data that users can access remotely via a computer network. One way to achieve this is through the use of server-based architecture that manages the ROM process as a part of the engineering design process. Central to this proposed architecture is the ROM server. As shown in Figure 2.2, the ROM server provides access to the ROM to all the producers and consumers of the engineering models. The ROM server is accessible over a computer network so ROM computations and analysis can be performed independently by all the consumers simultaneously without involving data copying and manual synchronization. The key activities the ROM server needs to provide are

- *Management and storage of the data*—A group of analysts can be involved in generating the computational snapshots required, based on the needs of the design engineers. This information is used as input to the ROM server, which manages all the computational data and ensures that it is available to other consumers as needed. This requires a mechanism that enables long-term persistence and storage of the data which enables users to access it when required. When an analyst updates this database, the ROM server needs to seamlessly trigger the computational processes

that recompute and update the necessary ROM parameters. The updated ROM parameters can then be made available to the engineers, analysts, and decision-makers involved in the engineering process.

- *Managing the ROM creation and update process*—After the snapshots for the ensemble matrix are computed the ROM server needs to collate the snaphots and compute the basis functions and coefficients for the reduced-order model. At this point the server should be ready and listening for client requests to perform ROM evaluations. When a producer updates the ROM database, the data is updated on the persistent storage immediately. However, the recompute of the ROM parameters must be carried out when no users are utilizing the ROM server resources. This is to ensure availability of the service during peak times when multiple engineers might be utilizing the ROM data for analysis.
- *Enabling interactive use and exploration using the ROM*—With data management, storage and the ROM creation processes in place, the constructed ROMs need to be usable by multiple engineers within the organization. Thus, being able to perform ROM evaluations based on user requested input is a critical component of the ROM server. In addition to computing the ROM for a given set of input parameters, the ROM server can also evaluate requests for the singular value spectrum, principal axes, and the projections of field variables on the dominant principal axes where each of these can be part of the analysis of the physical phenomenon being studied. Another integral piece of design space exploration is the ability to perform engineering optimization utilizing the ROM, which the ROM server must be capable of supporting with an appropriate client that incorporates the optimization algorithm.

Because all the data from the aforementioned analyses can be visualized, the ROM server can have the capability to create the outputs needed to support the design process.

- *Maintaining version control*—Many engineering organizations are geographically dispersed with engineers working across multiple time zones. Thus it is important that geographically distributed engineering groups have up-to-date access to the engineering data. When the ROM database is updated, the users automatically receive the updated model parameters from the ROM server, thus providing a unified version of all computational data across all the users in an organization. This ensures that once the model parameters are updated, all users have the exact same information. The ROM server is capable of synchronizing this information seamlessly, which eliminates the cost of explicit data synchronization between the users.

In the proposed engineering workflow the analysts and design engineers are better integrated into the overall engineering design process and can exchange information with greater ease. Real-time synchronicity enables the production and consumption of information independently and helps breakdown the sequential and unidirectional flow of information associated with batch processing.

2.3.1. Synchronization of data

Section 2.3 introduced the notion of producers and consumers of data and information in the context of the engineering design process. When a subset of the data changes due to an update from an information producer, other producers as well as consumers need to synchronize their copies of data to ensure that their engineering decisions are made based on the latest information. Thus when information needs to be exchanged between multiple producers and

consumers, a cost can be computed based on the information to be exchanged so that all the engineers have access to up-to-date information; this is the *cost of synchronization*, α. The cost of synchronization includes the costs of data exchange, computation, and the associated user costs. In this context, the data exchange cost refers to the time that is needed to move data from one computer to another (i.e., a larger volume of data leads to higher data exchange costs). Computational cost is the time taken to perform a computation on a specified dataset. It is a function of the type of computation and size of data. User cost is the communication time between users of engineering data for either requesting or notifying other users about an update to the data. Because this is an asynchronous form of communication between users, it includes the lag between the intent and the time the action is actually performed. The asynchronous forms of communication include but are not limited to email, phone conversations etc. In distributed engineering teams the user synchronization cost can be high and difficult to estimate. In this section several cases are outlined that determine the cost of synchronization between multiple producers and consumers considering various amounts of information exchanged when a ROM server is not used. Comparisons are then made to the corresponding cases when a ROM server is used to emphasize the decrease in user costs due to the utilization of a central data repository for engineering data and computations.

The types of data that need to be synchronized between a consumer and a producer using a POD-based ROM can be the entire ensemble matrix, the coefficients in conjunction with the basis functions, and the covariance matrix, each of which is essential for either the producer or the consumer to have the ability to evaluate an engineering model or make an engineering decision. Each has a different cost of synchronization and can be expected to follow the following inequalities

$$\begin{aligned} \alpha_{\mathbf{X}_{\text{cov}}} &< \alpha_{\mathbf{X}} \\ \alpha_{\mathbf{A}} &< \alpha_{\mathbf{U}} \end{aligned} \tag{2.4}$$

where $\mathbf{X}$ is the original ensemble matrix, $\mathbf{X}_{\text{cov}}$ is the covariance matrix, $\mathbf{A}$ is the coefficient matrix and $\mathbf{U}$ represents the basis functions. By synchronizing the covariance and coefficient matrices, $\mathbf{X}_{\text{cov}}$ and $\mathbf{A}$, respectively, the user can develop ROMs but can only access the initial snapshot models at a lower accuracy. On the other hand, having the entire ensemble matrix $\mathbf{X}$, which has a higher cost of synchronization, one can recreate the basis functions $\mathbf{U}$ and coefficients $\mathbf{A}$ in addition to accessing the initial snapshot models at the same accuracy that they were created at. Thus Eq. (2.4) signifies the trade-offs that can be made in order to balance the data exchange cost and accuracy of the ROM.

Furthermore, there can be multiple design iterations among the producers and consumers adding to the data synchronization cost because it requires that the information exchange be repeated among the producers and consumers. Thus, enabling synchronicity in design enables engineers to exchange information easily, makes design cycles shorter, and boosts the quality of designs by allowing deeper design space exploration. The remainder of this section examines the cost of data synchronization for each of these scenarios.

2.3.2. Producer-consumer synchronization

To develop the concept of synchronization cost further, we consider the case of a single producer and a single consumer, as shown in Figure 2.3a. The producer has the ability to modify computational models and generate the ROM, whereas the consumer utilizes the ROM to perform further analysis and model evaluation. The consumer can either (a) get the entire ensemble matrix $\mathbf{X}$ from the producer, (b) get a copy of the covariance matrix $\mathbf{X}_{\text{cov}}$ from the producer, or (c) get only the necessary coefficients $\mathbf{A}$ and basis functions $\mathbf{U}$.

a. In the case where the consumer receives a copy of the ensemble matrix, the coefficients and basis functions can be computed by the consumer and then used for ROM evaluations as needed. The cost of synchronization, α, is thus the time to synchronize the ensemble matrix between producer P_1 and consumer C_1 and the time to compute the ROM.

$$\alpha_{no-server} = T_{\mathbf{X}}^{P_1C_1} + t_{\mathrm{ROM}}^{C_1} \tag{2.5}$$

where T is the communication time and t is the consumer compute time.

b. Alternatively, because the size of the ensemble matrix $\mathbf{X}$ can be large, $N \times M$, the consumer can get a copy of the covariance matrix $\mathbf{X}_{\mathrm{cov}}$ of size $M \times M$ and compute the ROM parameters. In this case

$$\alpha_{no-server} = T_{\mathbf{X}_{\mathrm{cov}}}^{P_1C_1} + t_{\mathrm{ROM}}^{C_1} \tag{2.6}$$

The trade-off in this case is that the consumer does not have all the information about the computational model but has access to sufficient information to compute the ROM and examine the design space.

c. In this case the consumer requests only the ROM coefficients and basis functions, thus making the cost of synchronization, $\alpha_{no-server}$, a function of synchronizing the $\mathbf{U}$ and $\mathbf{A}$ matrices.

$$\alpha_{no-server} = T_{\mathbf{U}}^{P_1C_1} + t_{\mathbf{A}}^{P_1C_1} \tag{2.7}$$

Each of the above cases has inherent advantages and disadvantages in terms of the amount of data that needs to be synchronized as well as any additional computations. It should also be noted that in each of these cases as the number of design iterations increase, the synchronization factor increases linearly by a corresponding factor.

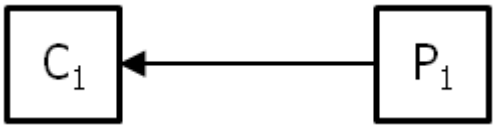

Figure 2.3a. Single producer and consumer information transfer.

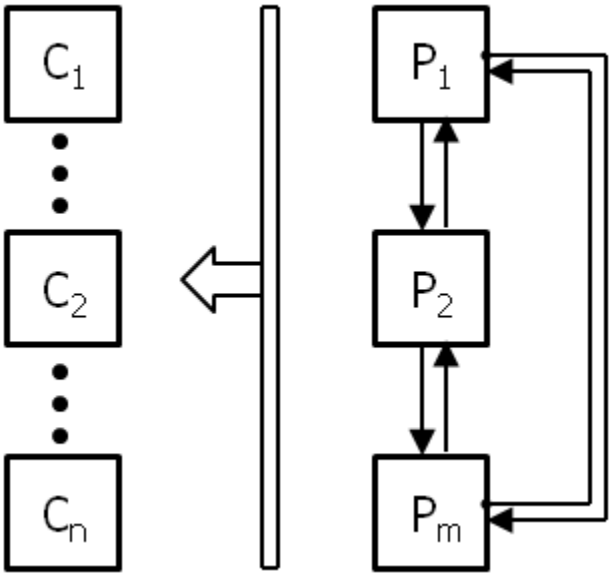

Figure 2.3b. Data synchronization between multiple producers and consumers.

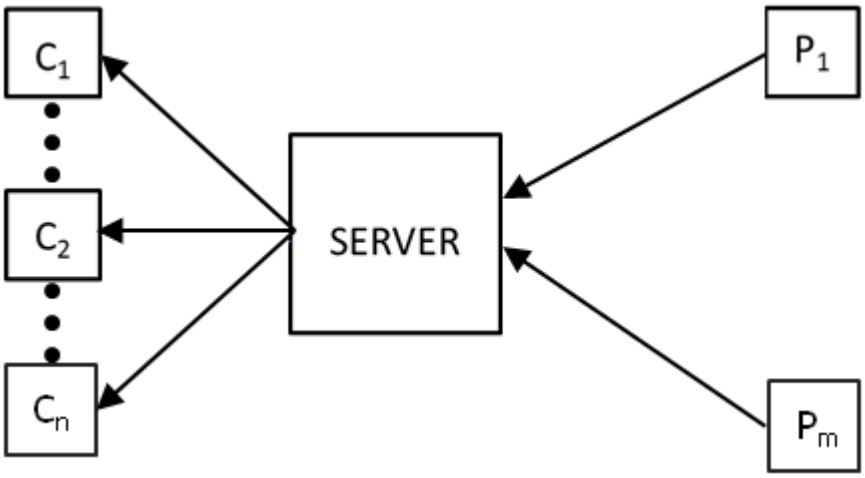

Figure 2.3c. Synchronization with ROM server.

2.3.3. Multiple producers and consumers

Synchronizing all information is more complicated when there are multiple producers $\{P_1, P_2, \ldots, P_K\}$ and multiple consumers $\{C_1, C_2, \ldots, C_L\}$ involved in the design process. This scenario is shown in Figure 2.3b. When there are K producers and each of the producers can make changes, there can be K concurrent changes to the ensemble matrix. Consider the scenario where L consumers have to get updates from one or more of the K producers. This is a two-step process which first requires a synchronization among all producers before synchronizing all consumers. For an update to be consistent with the global updates among the producers, all the producers must first synchronize their data. If $\mathbf{X}_C$ is the change to the ensemble matrix by any single producer, the total cost of synchronization for all the producers considering K changes. This is given by

$$ {}^{K}P_2 T_{\mathbf{X}_C}^{P_i - P_j} $$

where

$$ {}^{K}P_2 = \frac{K!}{(K-2)!} $$

Once this is done the ROM must be recomputed by each producer before a consumer can request an update from any one of the producers. Thus the cost of synchronizing all consumers, α is

$$ \alpha_{PC} = ({}^{K}P_2 T_{\mathbf{X}_C}^{P_i - P_j} + \sum_{i=1}^{L} T_{\mathrm{U+A}}^{P_i C_i} + \sum_{j=1}^{K} t_{\mathrm{SVD}}^{P_j}) \times R \qquad (2.8) $$

where R is the number of design iterations. Eq. (2.8) shows the high cost of explicit synchronization when there are multiple producers and consumers. The synchronization can be

performed implicitly using the ROM server, which reduces the cost of explicit user synchronization and is explained in detail in the following section.

2.3.4. Synchronization with a ROM server

In this section we study the cost of synchronization when a ROM server is utilized, as shown in Figure 2.3c. A major advantage of utilizing a server to synchronize and manage computational data is that producers do not have to perform explicit synzhronization. All data synchronization tasks are performed by the ROM server. Individual producers only have to notify the ROM server of updates that were performed. The ROM server then reconciles the data and manages updated versions of the ROM ensemble matrix. When these are updated, the ROM server also periodically computes the ROM basis functions and coefficients, thus maintaining a unified version of the computational data when accessed by clients.

The total cost of synchronization between any producer P_i and the ROM server R is

$$\alpha_P = \sum_{i=1}^{K} T_{\mathbf{X}}^{P_i R} \tag{2.9}$$

where $\mathbf{X}$ denotes a single update from a producer. Similarly the cost α_C for a consumer C_j to get data from the server is given by

$$\alpha_C = \sum_{j=1}^{L} T_{\mathbf{X}}^{RC_j} \tag{2.10}$$

Furthermore because the server recomputed the POD approximation periodically for updates to the ROM database, the total computational cost is given by

$$\alpha_{\text{POD}} = \sum t_{\text{SVD}}^{R} \tag{2.11}$$

Thus the total cost of synchronization is

$$\alpha_{server} = \alpha_P + \alpha_C + \alpha_{\text{POD}} \tag{2.12}$$

Comparing the equations for the synchronization cost in the case of multiple producers and consumers and utilizing the ROM server shows that

$$\alpha_{server} << \alpha_{PC} \tag{2.13}$$

This result supports the premise that the ROM server reduces the cost of synchronization between producers and consumers by managing the changes to computational data as well as performing the POD computations. Thus, repetitive computations are avoided and connectivity between engineers and analysts in the engineering team is enhanced.

In summary, when a ROM server is not used, there is a significant user cost associated with information synchronization in the engineering design process, whereas using the ROM server eliminates the need for explicit synchronization and hence reduces the user costs. A server-based solution has significant advantages when the engineering team is large and distributed globally. Producers and consumers of engineering models can work and exchange information simultaneously and in a planned manner. Furthermore, having a single repository for all the engineering models in an organization enables users to access the computations and analysis as needed.

2.4. Design of the ROM Server

Typically in scientific computing, when performing tasks over a network or utilizing multiple nodes on a shared memory machine, the emphasis has been on using programming tools like MPI and OpenMP to parallelize code and decrease the runtime. An MPI-based application typically runs on a cluster of computers distributing tasks over them and finally gathering the

results. OpenMP on the other hand is designed for shared memory architectures and best suited for single input multiple data (SIMD) problems (Quinn, 2005). MPI- and OpenMP-based computational solutions do not allow the results to be shared by several users simultaneously across a computer network. Moreover, these compute jobs are typically batch processes (i.e., they run once and have to be rerun for the next numerical experiment). In contrast, the server-based solution needed here involves a server process running continuously in the background and is always available to respond to client requests for ROM evaluations as well as edits to the ROM database. This enables multiple engineers to simultaneously leverage and share the same data while performing independent tasks and computations. For example, one design engineer could be analyzing a flow-field while another engineer simultaneously works on an optimization problem based on the same data, both without local access to the entire ROM database. Furthermore, the server-based ROM database solution helps provide vendor agnostic access to data, computations, and analysis. Specifically, although there are a number of commercial packages available to build engineering models, the ROM server as implemented here has the ability to read in data from most commercial packages utilizing the VTK data format (VTK, 2015). This gives the server the ability to process and compute ROMs from multiple data sources. In addition, users when accessing the computations and analysis via the ROM server need not have access to the commercial package that was used to generate the initial set of models, thus helping provide vendor agnostic access to engineering models and analysis to experts and non-experts alike.

Thus to get these benefits, the underlying architecture for the ROM server is client-server based. The main purpose of the ROM server is to be available to respond to user requests and perform the required computations in a timely fashion. Because multiple users can be requesting

evaluations simultaneously, the server must have a thread pool capable of processing such requests. There are four key building blocks needed:

- Create and manage a computational database that is accessible over a computer network (TCP/IP)
- Create the ROM
- Enable a client node to submit an evaluation request that executes on the ROM server and sends the results back to the client.
- Enable the ROM server to detect additions and deletions of computational data to the database and automatically schedule recomputations of the POD basis functions.

Figure 2.4. shows the main components of the ROM server architecture. The dotted lines indicate loading of the POD databases from disk into memory on startup and the solid lines indicate operations of the server after startup. This must be achieved in a manner such that each user gets the correct information back regardless of the number of concurrent users. If there are edits to the ROM database, these should be reflected in the computations in a timely manner, so that stale information is not used to make engineering decisions on critical components. To achieve fast computations, the server on startup loads the coefficient $\mathbf{A}$ matrix and basis function $\mathbf{U}$ matrix into memory. From then on, as long as the server is running, all POD evaluations are done using the information stored in memory using these matrices and Eq. (2.1). The design considerations of the ROM server are

- A schema has to be developed to store the computational models and design parameters associated with them on disk.

- For the client and the server to communicate, they both need to be able to interpret data passed from one to another correctly. This requires the construction of communication protocols for the messages that need to be passed between them.
- The server must be capable of accepting client connections and making ROM evaluations as per their requests. This requires the selection of an appropriate data transport layer, in this case TCP/IP. The server must also be capable of accepting multiple client connections from analysts and design engineers, and processing them appropriately. This requires a pool of threads that can coordinate with one another. Because the threads in the thread pool share the same process address space, multiple threads can perform the same operation simultaneously on shared data structures which leads to race conditions (Quinn, 2005). Hence adequate precautions should be taken to enable data sharing while avoiding race conditions.
- The need for the server to recompute the POD basis functions automatically requires a background thread that cannot be invoked by a client. For this purpose a background thread has to be created, and a suitable algorithm has to be developed to enable time-triggered recomputations.

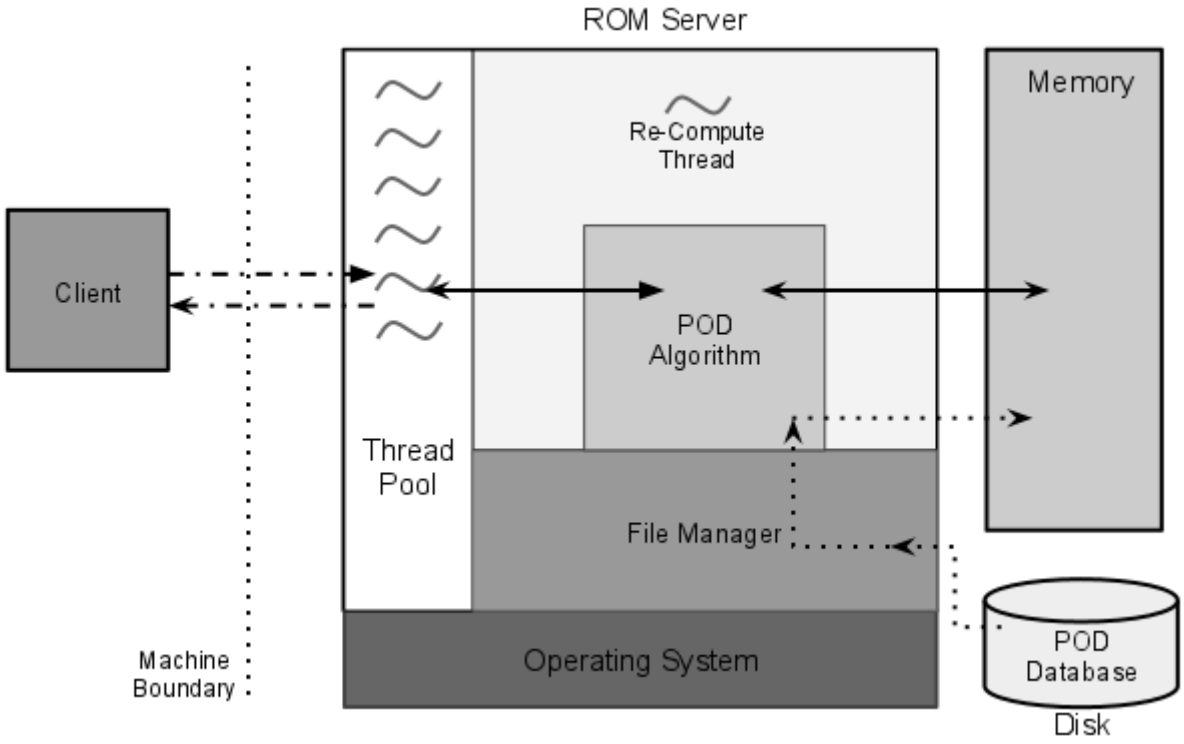

Figure 2.4. Main components of the ROM server architecture.

In this work TCP/IP is used as the transport medium for communications between the client and the server (Kozierok, 2005). The primary reason is that TCP/IP guarantees ordered delivery of packets, which is critical in this application. Remote procedure calls are used to establish communications between the client and server at the application layer of the networking stack (Kozierok, 2005). It must be noted that in this case the availability of the ROM server is on a best-effort basis, and when recomputing the POD basis functions, it is briefly unavailable for client requests. This design trade-off is explained in detail in Section 2.4.2.

2.4.1. Server thread pool

For a server application it is important that it have the ability to handle multiple client requests simultaneously. In the ROM server, this requirement is handled by having a thread pool containing multiple threads of execution. When a request for information or performing a computation comes in from a client, a free thread from the thread pool is assigned the task, performs the requested task, and sends the information back to the client. Once it is done with this task, the thread marks itself available and waits until another client request is assigned to it. In the example developed here, the ROM server has a total of 10 threads to process client requests in addition to the main thread of execution. The number of threads in the pool is configurable. It was chosen by trying to balance the requirement for multiple client requests and the total number of processors available on the machine, in this case 16. If more processors are available the thread pool size can be increased as needed.

2.4.2. ROM recomputations

During the engineering design process situations can arise where the design parameter space increases, requiring that additional computational models be added to an existing database. Similarly some design parameters can be deemed unnecessary, which can lead to discarding

existing computational models. These scenarios have to be accounted for with minimal disruption to the continual operation of the ROM server to make it an effective tool for managing computational data. Thus, while the ROM server is running, edits should be allowed to the database of computational models. These edits can be datasets that have been added or deleted from the ROM database.

Such edits require recomputation of the POD basis functions and coefficients to update POD evaluations to reflect the new information. The process of updating the POD basis functions leads to multiple software design choices to schedule the recompute task, each of which are briefly explained:

- *Immediate scheduling* in which recomputation is scheduled based on the availability of new information. Immediate scheduling has the inherent advantage of making new information available immediately; however, there exists the possibility that there may be multiple consecutive changes to the ROM database in which the server performs repeated POD basis function recomputations, each superseding the previous one. This creates unnecessary delays in the work and results in unnecessary compute cycles.
- *Delayed scheduling* in which recomputation waits for periods of no client activity to incorporate new data in the ROM database. Delayed scheduling has the advantage that existing requests for computations are given high priority. The primary disadvantage of delayed scheduling is that during periods of prolonged activity all client requests get stale information, and it requires manual coordination among the clients to cease new requests in order to trigger a POD basis function update.

- *Periodic scheduling* in which recomputation of the POD basis functions is scheduled at a regular time interval that can be configured as per the requirements of the engineering team. The major advantage of periodic scheduling is that multiple changes to the ROM database can be reconciled and computed at once without utilizing compute cycles for each update. This also establishes a deterministic schedule and ensures availability of the same information to the entire engineering team at all times. Thus the producers can work to have the updates done before the recompute, and the consumers know when to expect the next update. The tradeoff in this method is that updates to the ROM database may not be immediately available for use. In the example that follows the authors have chosen this scheduling technique due to its predictable nature and its advantage of being configurable to the work habits of the engineering team.

To accomplish the task of computing the POD basis functions and coefficients while the ROM server is running, a background recomputation thread has been incorporated in the design of the server. This thread is a lazy asynchronous timer thread that checks the ROM database for changes at a specified frequency. Only when an edit is detected does this thread proceed with the recomputation. This is a background thread and does not belong to the thread pool discussed in the previous section, which processes client requests. This thread is in the idle state for most of the time and is alerted by an operating system signal at the end of the specified time period. Once it is out of the idle state, it proceeds to check the timestamp of the last ROM database update and compares it with its own copy from the previous time it was alerted. If a ROM database update has occurred since the previous run, it proceeds to perform a recomputation. Because recomputation of the POD basis functions is a computationally expensive task (i.e., it can take

several minutes depending on the size of the ensemble matrix), this thread should be scheduled to run during periods of no utilization of the ROM server.

Furthermore once the POD basis functions are computed, client requests for POD evaluations are not processed for the brief period while the data structures are being replaced in memory. Although it is possible to serve existing client requests with the older version of the ROM, we chose not to do so to ensure that all clients get access to the same information once a recomputation is complete. The time-based triggering algorithm is described in pseudo code as follows:

```
Algorithm (1)
Parameters : Database update time (tdb), Previous run time of thread (tp)
Procedure ( Re-computation )
Step 1 :    If ( tdb > tp )
            Re-compute POD
            Close all existing client connections
            Stop listening to newer client requests
            Update in-memory data structures
            Re-open server to process client requests
            Sleep
      Else
            Sleep until next alert
            Goto Step 1
End Procedure
```

2.5. Application to Heat Exchanger Fin Shape Design

Section 2.4 described the design of the ROM server; this section provides an example in which multiple engineers exchange computational models to perform a collaborative engineering task using a ROM server. The various interactions are studied in the absence of and then in the presence of a ROM server. The scenario involves two data producers (P_1 and

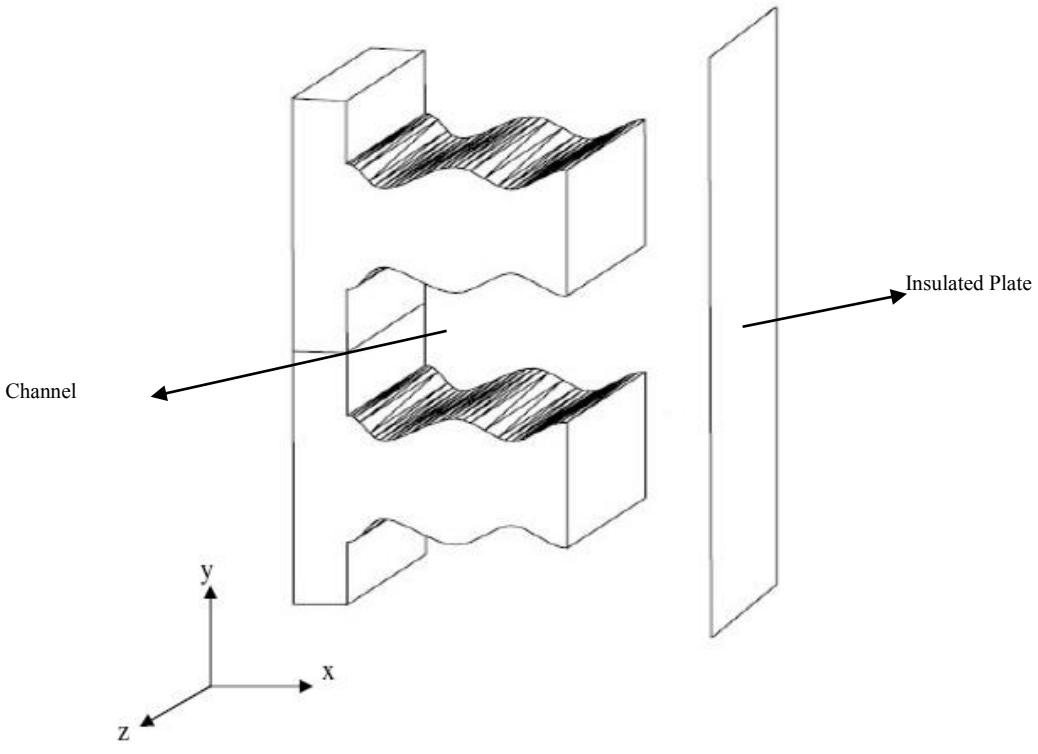

Figure 2.5. Schematic diagram of the fins.

P_2) and three data consumers (C_1, C_2, and C_3). The producers in this case study are responsible for generating computational models, and the consumers perform engineering tasks related to analysis and optimization. Yet another important requirement is that the producers as well as the consumers should have access to all the models as well as the analysis that has been performed by the team as a whole. It should be noted that the focus of this work is on studying the interactions between the producers and consumers and not on the actual analysis of the problem under consideration. A two-dimensional heat exchanger fin shape design problem has been chosen as the engineering design problem in which multiple engineers work on different aspects of the overall design. The designs are then used to create a ROM. This ROM is utilized for further analysis of the design.

2.5.1. Mathematical model and numerical solution

Heat exchanger fins are used to enhance removal of heat from any heated surface. Heat exchanger fins are of particular importance in cooling electronic components and other machinery and equipment. Changes in the shape of the fin can result in improved performance, thus reducing cost, space needed, and energy required for fans (Kays and London, 1998). A number of researchers have studied the problem of heat dissipation from longitudinal fins, and earlier the authors developed an evolutionary algorithm coupled with numerical simulations to optimize the shape of heat exchanger fins (Suram, Ashlock and Bryden, 2006), to which the reader is referred to for a more detailed discussion of this design problem. This section describes in brief the governing equations of momentum and energy as applied to the heat exchanger fin model, the numerical solution, and development of the ROM.

Figure 2.5 shows a set of fins. Fluid (water) is pumped through the channel between the curved surfaces of two consecutive fins to remove heat. The direction of flow is along the

positive *z*-axis. It is assumed that the flat plate is insulated and that the fluid velocity and temperature profiles are fully developed. The flow is also assumed to be laminar and incompressible, and any effects of natural convection are neglected. The whole system is assumed to be in steady state. Taking advantage of the symmetry, only one half of the fin is modeled, as shown in Figure 2.6. The distance from the base of the fin to the insulated flat plate is assumed to be of unit length. The length of the fin is denoted by *a*, the base thickness by τ and the spacing between two consecutive fins by *2b*. The shaded portion represents the lateral surface of the fin. The thermal properties of the solid and fluid are assumed constant and only quadratic fin profiles are considered in this shape design problem. The mathematical model describing the fluid flow is given by

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = \frac{1}{\mu}\frac{\partial p}{\partial z} \tag{2.14}$$

The temperature distribution in the fin is

$$\frac{\partial^2 \theta_s}{\partial x^2} + \frac{\partial^2 \theta_s}{\partial y^2} = 0 \tag{2.15}$$

and the temperature distribution in the fluid is given by

$$\frac{\partial^2 \theta_f}{\partial x^2} + \frac{\partial^2 \theta_f}{\partial y^2} = \frac{\rho_f c_{pf}}{k_f} v \frac{\partial \theta_f}{\partial z} \tag{2.16}$$

The momentum equation has no-slip at the fluid-solid interfaces and symmetry boundary conditions on the other surfaces, respectively. The energy equations also have symmetry

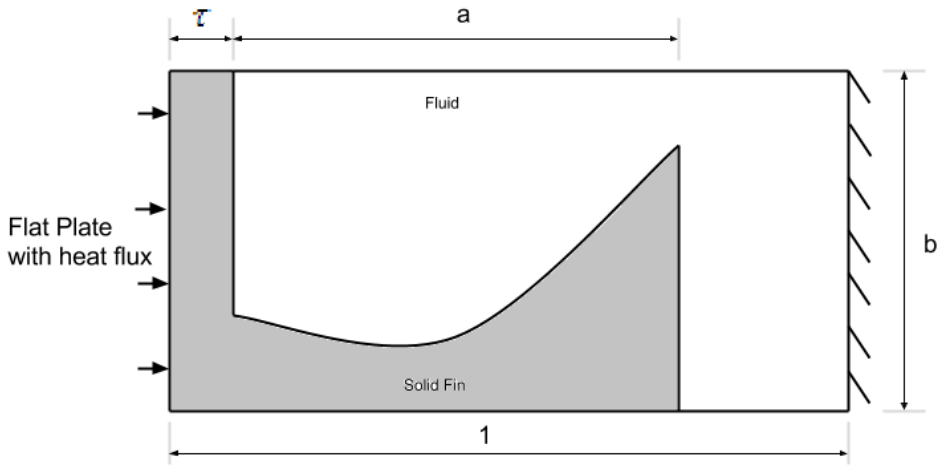

Figure 2.6. Single fin being modeled.

Table 2.1. Representative shape designs.

Producer	Base Thickness (τ)	Fin Spacing (b)	Points on Lateral Surface			Shape
			y_0	y_1	y_2	
P_1	0.118	0.224	0.126	0.115	0.210	
P_1	0.011	0.334	0.326	0.115	0.210	
P_1	0.020	0.134	0.086	0.085	0.075	
P_1	0.200	0.219	0.186	0.103	0.052	
P_1	0.05	0.315	0.280	0.203	0.251	
P_2	0.150	0.250	0.050	0.100	0.210	
P_2	0.181	0.270	0.250	0.100	0.189	
P_2	0.030	0.450	0.360	0.320	0.360	
P_2	0.121	0.420	0.213	0.312	0.213	
P_2	0.142	0.320	0.183	0.183	0.183	

boundary conditions on the symmetric surfaces. At the solid-fluid interface, the boundary condition must satisfy the requirements of equality of temperatures and temperature gradients. The above governing equations were solved numerically using the finite difference method. Further details on the discretization, grid generation, and numerical solution can be found in (Suram, Ashlock and Bryden, 2006).

2.5.2. Shape design and ROM construction

The objective of the fin shape design problem discussed here is to study the velocity field and temperature distribution when the shape of the lateral surface of the fin is varied. With the temperature and velocity profiles known, the heat transfer coefficient, the pumping power, material cost, and manufacturing cost can be determined and considered as a part of the design space. In this study the base thickness of the fin and spacing between the fins can be varied in addition to the lateral surface being constrained to being quadratic in shape. A total of 15 designs chosen by sampling the allowable design space were used as the initial snapshot set. Table 2.1. shows some representative designs created by producers P_1 and P_2. The shape design problem has five parameters, and the initial set of designs was chosen to account for all possible curvature variations of the lateral surface of the fin. The base thickness and fin spacing were chosen by randomly sampling the range of allowable values. Figure 2.7 shows the distribution of the design space covered by the initial set. In Figure 2.7, to represent the designs in three dimensions, the average of the three values of the points on the lateral surface of the fin is used in place of the actual values. It is possible that the sampling of the initial design set propagates error into the ROM, which can be overcome by adding and removing models as necessary from the ROM server. Also, if the initial snapshot set is insufficient to build an accurate ROM, more designs can be added to the ROM server as needed. A concatenated ensemble matrix with the temperature

and velocities at each point in the discretized domain was constructed. The energy of the POD expansion is used to compute the accuracy of the ROM as described in Eq. (2.3). If the accuracy of the ROM is outside the acceptable range, more models are added to the snapshot set. To simulate the design process and the exchange of information between the producers and consumers, the following sequence of operations is adopted:

1. 15 designs were computed and used as the initial input to the ROM server
2. Consumers make use of the ROM server to evaluate designs and analyze data
3. More designs are added to the ROM server by the producers
4. All consumers seamlessly get updated information without making explicit updates and are able to evaluate designs that were previously not possible.

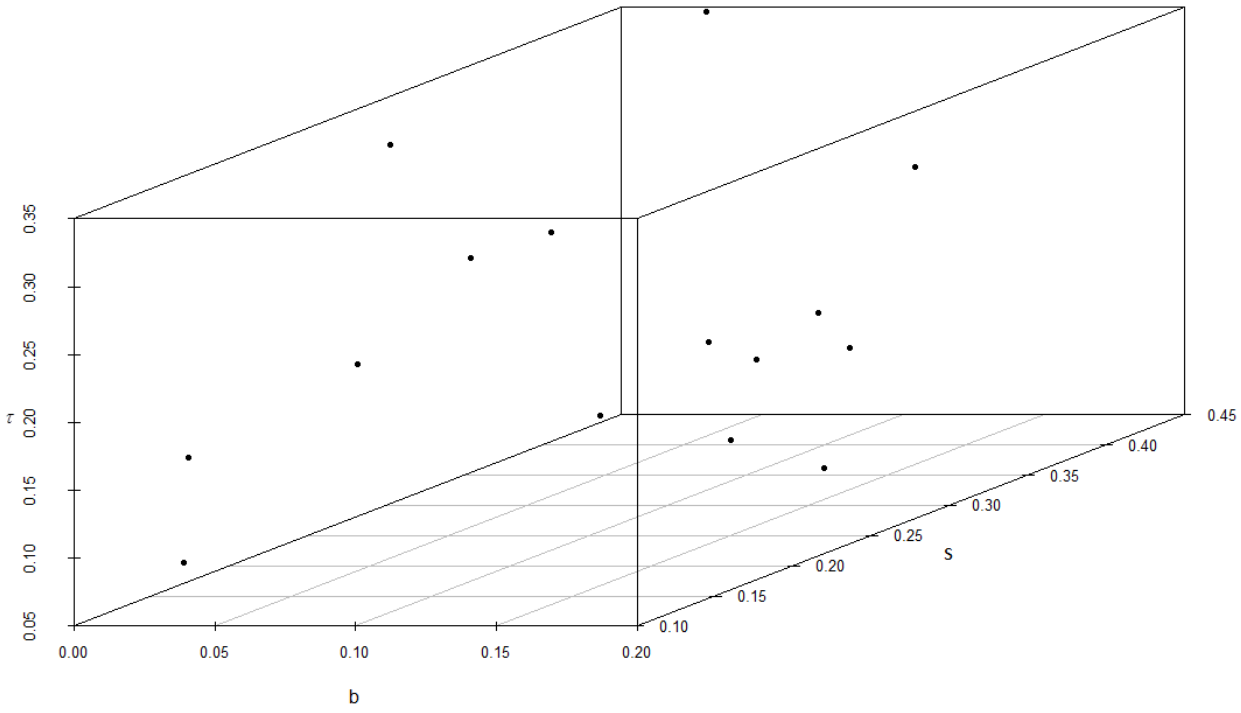

Figure 2.7. Initial design space of heat-exchanger designs.

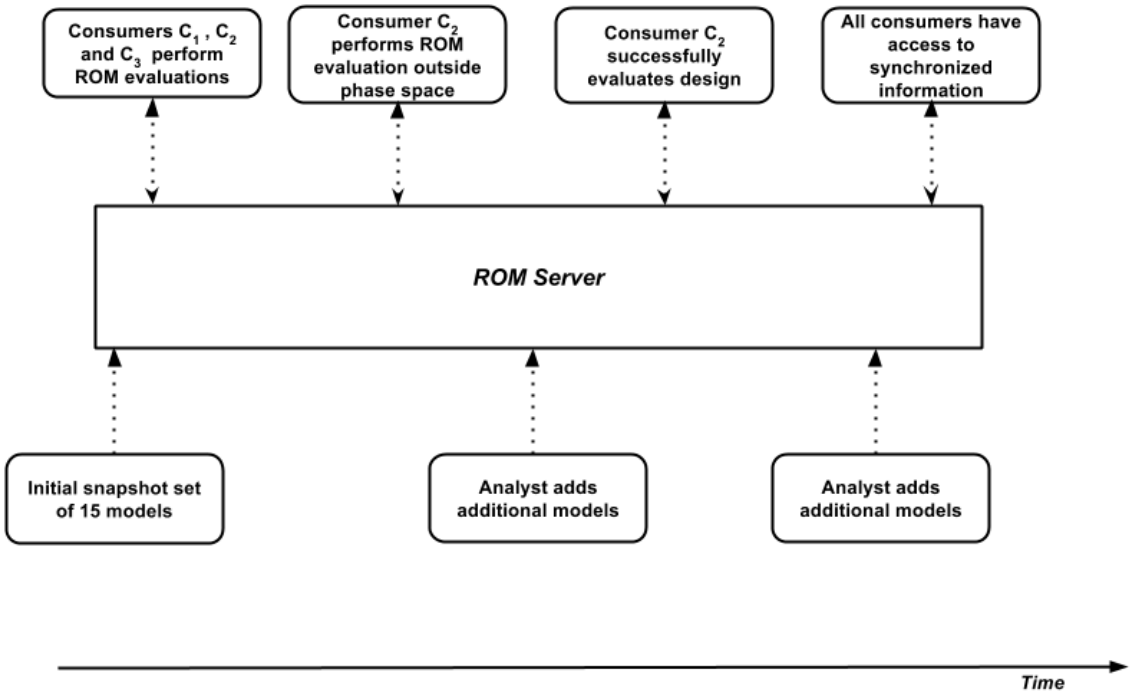

Figure 2.8. Timeline of various producer consumer interactions during the design process.

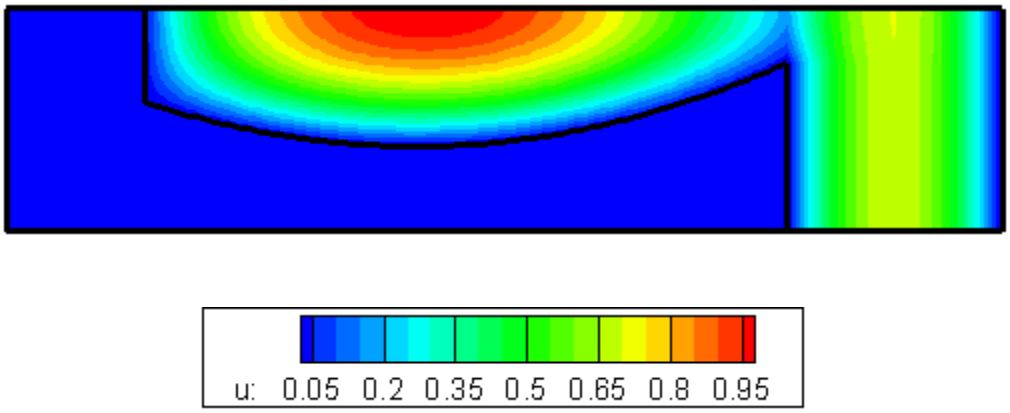

Figure 2.9a. Velocity profile.

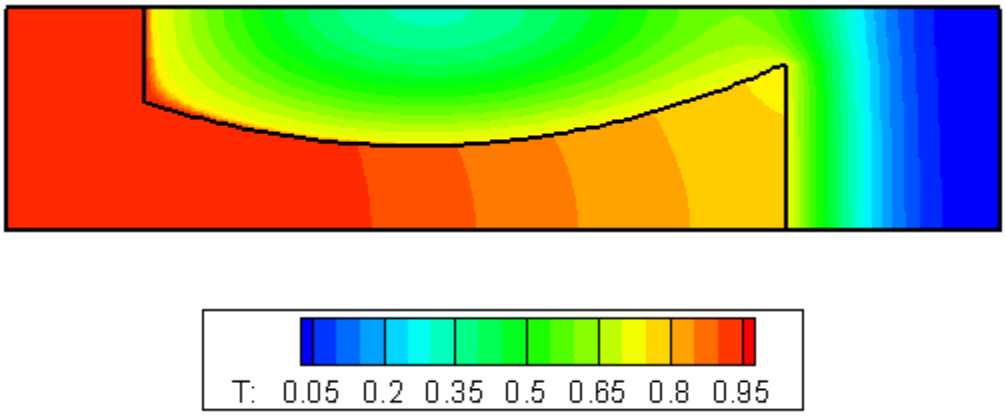

Figure 2.9b. Temperature profile.

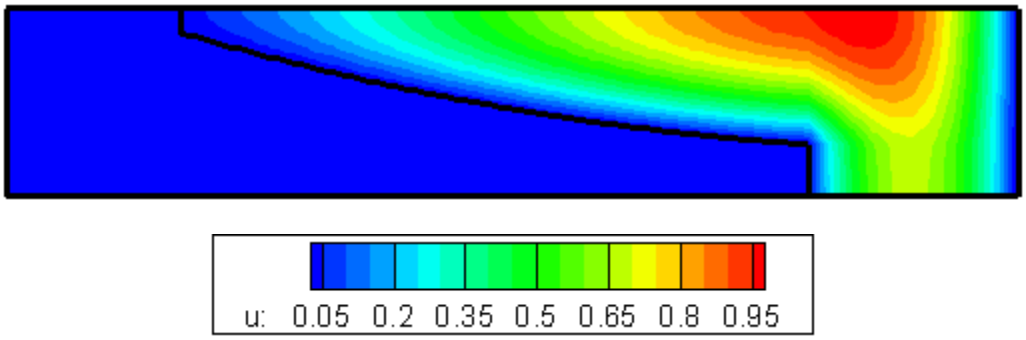

Figure 2.10a. Velocity distribution.

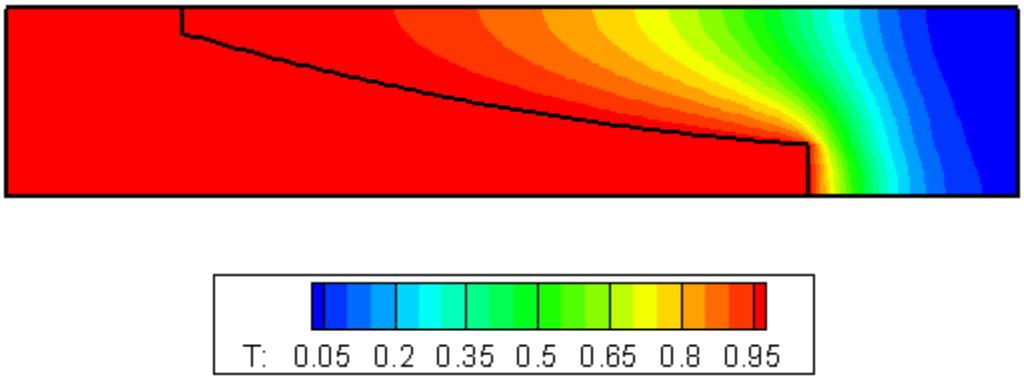

Figure 2.10b. Temperature distribution.

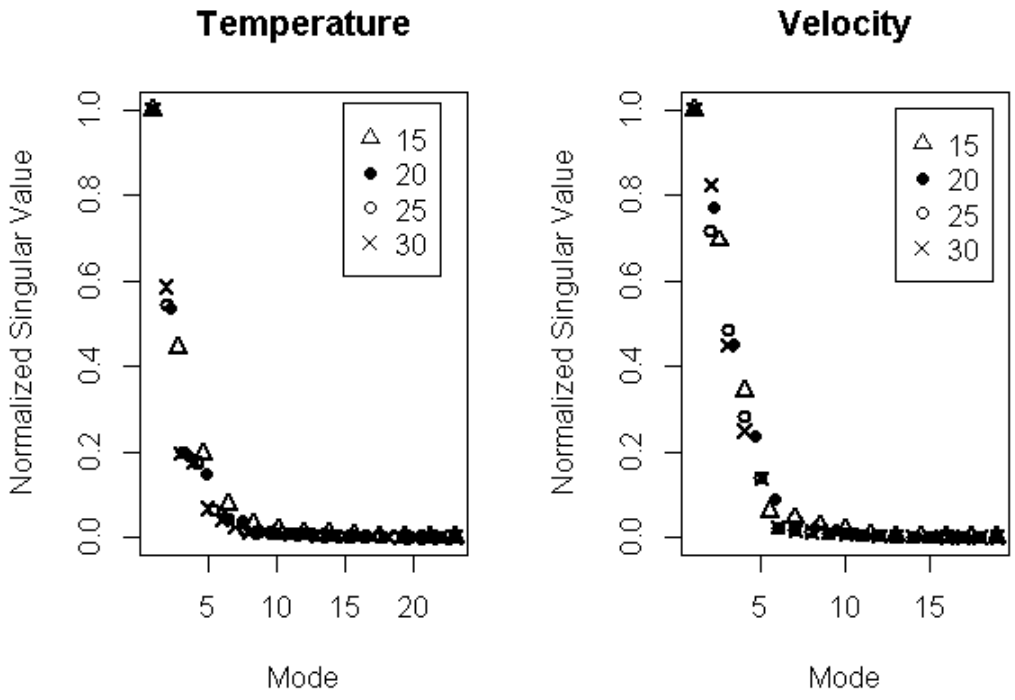

Figure 2.11. Variation of singular value spectrum with number of computational models.

2.5.3. Information exchange and the ROM server

The ROM server eliminates the need for explicit information synchronization between producers and consumers during the design process. This section extends in detail the design process outlined in Section 2.4.2. The steps in the design process are simulated as shown in Figure 2.8.

1. The producers P_1 and P_2 run the computational experiments needed to generate models for the POD ensemble matrix by varying the design parameters that are used to construct the ROM. Initially the ROM database holds the results from 15 computational models developed by the producers. Thus after this step the ROM server manages a database of 15 computational models, and the design space encompassed by this set of models is available to consumers (clients) to utilize for various tasks.
2. Evaluations of the ROM for design parameters not in the initial set of computational results are computed by the consumers, who also have the ability to request individual modes as well as the singular value spectrum from the ROM server. A consumer C_1 uses the ROM server to evaluate the velocity and temperature profiles for the following design parameters, which though not in the initial solution set lie within the design space of the problem. The input to the ROM server is the following list of parameters:

	b	y_0	y_1	y_2
0.160	0.260	0.150	0.100	0.195

The ROM server then performs the computation and returns the results to consumer C_1 as shown in Figure 2.9.

3. Another consumer, C_2, attempts to perform an evaluation for the following set of design parameters:

	b	***y0***	***y1***	***y2***
0.210	0.220	0.190	0.105	0.060

This request for evaluation of a proposed design is not within the design space of the existing ROM based on the 15 computational models. This results in a non-evaluation and the return of an appropriate message to consumer C_2 and a notification to producer P_2 that additional analysis is needed to extend the design space. This may lead to a discussion between the consumers and producers about the direction of design, the design space, and the analysis goals.

4. Producer P_2 and consumer C_2 consult on the needed expansion of the design space, and producer P_2 performs the additional analysis and adds it to the ROM server. In some cases, for example, due to physical constraints it might not be possible to extend the design space to accommodate the request from consumer C_2.
5. The ROM server recomputes the ROM.
6. Producers P_1 and P_2 continue to populate and extend the design space with more models to the ROM server. The consumers can utilize the new information without

explicit synchronization of the data. For example, the unsuccessful attempt to evaluate the model in step 3 can now be evaluated after additional models have been added to extend the design space. Thus the consumer C_2 can now request the ROM server to perform the same computation and visualize the normalized flow and temperature fields, as shown in Figure 2.10.

7. Just as the POD-based ROM can be evaluated to determine the performance of a particular design configuration, consumers can also get information associated with the ROM, for instance the spectrum of singular values. The singular value spectrum shows the amount of energy associated with each dominant mode and is hence indicative of the number of modes to be considered in the POD approximation. Figure 2.11 shows the variation of the singular value spectrum with the number of computational models utilized by the ROM server for the POD approximation. It is seen that as more computational models are added, the singular values begin to converge. In this case adding more than 15 computational models did not increase the accuracy of the ROM because the incremental energy captured by the later modes is only a small fraction of the total energy of the POD approximation. In this case the first five, ten, and fifteen modes capture 89.7%, 98.6%, and 99.99% of the total energy, respectively. Thus a ROM approximation based only on the first five modes will have a larger error compared to a ROM computed using the first fifteen modes. For a design engineer using a ROM to make engineering decisions, the singular value spectrum is an indication of the error (i.e., if the error is higher than the maximum acceptable value, more data needs to be added to the snapshot set for the ROM server to recompute an updated model). Because full CFD simulations are performed at the

end of the design phase before finalizing a design, an appropriate cutoff of the number of modes can be made to enable design decisions at the early stages in the process.

By organizing and managing computational models in the ROM server, all members of an engineering team have access to the same information without repeated manual synchronization of data. It has also been demonstrated that each of the design tasks in the workflow can be accomplished using the POD-based ROM server, and the system helps all consumers seamlessly get updated information without making explicit updates. This results in the organization saving time as well as serving the purpose of maintaining the version history of computational models.

2.6. Conclusions and Future Work

In this work a computational data management system is proposed that incorporates reduced-order modeling to enable a distributed framework for engineering design. The data management system has been enabled by developing a client-server based architecture. The server stores computational data in a vendor agnostic format and enables POD computations that can be used in analysis and optimization for simulation-based engineering design. The client-server based architecture enables persistent storage of computational data that can be accessed on-demand from any geographical location, thus improving collaboration among members of a distributed engineering team. The cost of synchronization was computed based on the size of the data and the number of times it needs to be transferred amongst multiple producers and consumers participating in a product design cycle, and this cost was computed for the cases with and without the ROM server. It was found that the cost of synchronization is lower when a ROM server is used in the design workflow. The software infrastructure developed allows real-time

collaboration in a distributed engineering team by leveraging the fast computational capabilities of POD-based ROMs. As discussed above a batch process creates a bottleneck to information sharing, which is overcome by utilizing a server-based solution. Furthermore, the client-server architecture enables exposing a synchronized view of all existing computational models and also accounts for the changes to these models in a near real-time fashion. The centralized server-based solution also eliminates the user cost of data synchronization, which is high in a distributed engineering team. The solution developed in this work is especially useful to help with collaboration in a geographically distributed team of analysts and design engineers.

Further research is needed to address several issues. To assemble the ensemble matrix in the POD algorithm, all the CFD datasets are required to have the same number of grid points. This is a rather stringent requirement in most cases. Developing an algorithm to reconcile data from CFD models into the ensemble matrix can help automate construction of the ensemble matrix. In the client-server model because the computation is being performed on a remote server, the total computational time is the sum of the time to perform the arithmetic and the network communications time. The network communication time introduces latency, which was not considered in this article. This is an important aspect and needs further study. This current work, implicitly requires that the computational database be located on a single node because of the restrictions imposed by the ROM server. In the future the authors plan to explore the possibility of utilizing a cloud-based ROM database to store as well as compute ROM parameters. The capabilities of the ROM server can thus be extended to incorporate multiple computational databases simultaneously. This will require that the ROM computation be performed in a distributed manner, where each node in the network can access data on other nodes based on some global metadata. Another aspect of this work that can be made more

efficient is the recomputation of the POD basis functions when new data is added or removed from the ROM database. This can be improved by utilizing SVD update methods so that the ROM server is always available for processing requests. A lazy recomputation can be performed when the ROM server has a negligible client load. Another aspect of this work that needs further study is the development of appropriate clients that access and leverage the ROM server. For instance, an optimization client can be developed that utilizes the ROM server to evaluate the objective function. Client-side caching techniques can be developed to minimize the effects of latency in making network calls. Also, versioning of engineering models can help engineers keep track of and compare incremental changes during the design process.

Acknowledgement

This research was supported in part by the US Department of Energy – Office of Fossil Energy under Contract No. DE-AC02-07CH11358 through the Ames Laboratory.

References

Tanenbaum A., Steen M., 2002. Distributed Systems: Principles and Paradigms. Prentice Hall.

Kirby M., 2001 Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns. Wiley.

Suram S., McCorkle D., Bryden K., 2008. Proper orthogonal decomposition based reduced order model of a hydraulic mixing nozzle. AIAA MAO, Vancouver, Canada.

Kays W.M., London A.L., 1998. Compact Heat Exchangers, 3rd ed, Krieger.

Samadiani E., Joshi Y., 2010. Reduced order thermal modeling of data centers via proper orthogonal decomposition: a review. International Journal of Numerical Methods for Heat & Fluid Flow, Vol. 20 Iss: 5, pp.529 – 550.

Trefethen L.N., Bau III D, 1997. Numerical Linear Algebra. SIAM, Philadelphia.

Meyer C.D., 2000. Matrix Analysis and Applied Linear Algebra. SIAM, Philadelphia.

Tan B.T., 2003. Proper Orthogonal Decomposition Extensions and Their Applications in Steady Aerodynamics. MS Thesis, Singapore-MIT Alliance.

Bui-Thanh T., Damodaran M., Willcox K., 2004. Aerodynamics Data Reconstruction and Inverse Design using Proper Orthogonal Decomposition. AIAA Journal, vol. 42, No. 8.

Everson R., Sirovich L., 1995. Karhunen- Loève Procedure for Gappy Data. Journal of the Optical Society of America, 12(8), 1657-1664.

My-Ha D., Lim K.M., Khoo B.C., Willcox K., 2007. Real-time optimization using proper orthogonal decomposition: Free surface shape prediction due to underwater bubble dynamics, Computers and Fluids, Vol 36, Issue 3: 499-512.

VTK, www.vtk.org, accessed on 1/27/2015.

Astrid P., 2004. Reduction of Process Simulation Models: a proper orthogonal decomposition approach. PhD Thesis, Technische Universiteit Eindhoven.

Seymour K., Yarkhan A., Agrawal S., Dongarra J., 2005. NetSolve: Grid Enabling Scientific Computing Environments In Grid Computing and New Frontiers of High Performance Processing, vol. 14, Advances in Parallel Computing.

Kozierok C.M., 2005. The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference. No Starch Press. 1st ed.

Tannehill J.C., Anderson D.A., Pletcher R.H., 1997. Computational Fluid Mechanics and Heat Transfer. 2nd ed.. Taylor & Francis.

Mahule T et al., 2010. PADMINI: A Peer-To-Peer Distributed Astronomy Data Mining System and a Case Study. Conference on Intelligent Data Understanding.

Djilali S., 2003. P2P-RPC: Programming Scientific Applications on Peer-to-Peer Systems with Remote Procedure Call. Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03), pp.406. CCGrid.

Hunter J., Choudhury S., 2005. Semi-automated preservation and archival of scientific data using semantic grid services. Fifth IEEE International Symposium on Cluster Computing and the Grid, vol. 1, pp.160 - 167. CCGrid.

Sakurai T. et al., 2006. A Hybrid Parallel Method for Large Sparse Eigenvalue Problems on a Grid Computing Environment Using Ninf-G/MPI. Large-Scale Scientific Computing. pp 438-445, Springer.

Gunzburger M.D., 2002. Perspectives in Flow Control and Optimization. Society for Industrial and Applied Mathematics.

Kerschen G., Golinval J.C., Vakakis A.F., Bergman L.A., 2005. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems. Nonlinear Dynamics, vol. 41, pp 147-169.

Ly H.V., Tran H.T., 2001. Modeling and control of physical processes using proper orthogonal decomposition. Mathematical and Computer Modeling, vol. 33, pp. 223-235.

Chen J., Kostandov M., Pivkin I.V., Riskin D.K., Willis D.J., Swartz S.M., Laidlaw D.H., 2009. Visual analysis of dimensionality reduction for exploring bat flight kinematics in a virtual environment. Joint Virtual Reality Conference of EGVE - ICAT - EuroVR.

Zhou X., Hitt D.L., 2011. Proper Orthogonal Decomposition Analysis of Coherent Structures in Simulated Reacting Buoyant Jets. AIAA Journal, vol. 49, issue 5, pp. 945-952.

Korpela E., 2011. Distributed Processing of SETI Data: Searching for extra terrestrial intelligence. The frontiers collection, Part 2, pp 183-199.

Liu J., 2003. Micro-benchmark level performance comparison of high-speed cluster interconnects. Proceedings of the 11th Symposium on High Performance Interconnects.

Ghia U., Ghia K.N., Shin C., 1982. T: High-Re Solutions for Incompressible Flow Using the Navier-Stokes Equations and a Multigrid Method. Journal of Computational Physics, 48, pp 387-411.

Du, J., Zhu, J., Luo, Z. and Navon, I. M., 2011. An optimizing finite difference scheme based on proper orthogonal decomposition for CVD equations. International Journal for Numerical Methods in Biomedical Engineering, 27: 78–94.

Pahl, G., Beitz, W., Schulz, H.-J., Jarecki, U., Wallace, K.B., Lucienne T.M. (Eds.), 2007. Engineering Design: A Systematic Approach, 3rd Edition, Springer Verlag.

Ertas A., Jones J.C., 1996. The Engineering Design Process 2nd ed, Wiley.

Suram S., Ashlock D.A., Bryden K.M., 2006. Graph based evolutionary algorithms for heat exchanger fin shape optimization, Portsmouth, VA, United states: American Institute of Aeronautics and Astronautics Inc., 647-657.

Quinn M., 2005. Parallel Programming in C with MPI and OpenMP, 1st ed, McGraw-Hill.

CHAPTER 3. A DISTRIBUTED SYSTEMS APPROACH TO ENGINEERING MODELING

Article to be submitted to *Advances in Engineering Software*

Sunil Suram, Nordica A. MacCarty

and Kenneth M. Bryden*

Abstract

In this paper we present a novel methodology for modeling engineered and other systems based on integrating a set of component models that are accessible as "model-as-service" components within a cloud platform. These component models can be combined together to form a systems model. The component models are stateless and web-enabled. The advantage of being web-enabled is that developers can use the models as API endpoints as opposed to library components, hence making the models themselves language agnostic and less restrictive in their use. These ideas are presented within the context of a previously published engineering model for the preliminary thermal analysis and design of a small biomass cookstove. In this paper the monolithic biomass cookstove model is separated into six independent, stateless component models supported by a generic model application infrastructure. Interaction between the models is orchestrated by a federated model system. Finally, the efficiency of the cookstove from the monolithic model was compared with the distributed systems model, orchestrated by the federation management system. It was found that there was no change in the efficiency. However, the systems model increased the time-to-solution due to network latency. In conclusion, it is advantageous to build web-enabled component models for their easy reuse

across multiple systems models. Furthermore, if the computational time of a model is high, the effects of network latency can be neglected, because a model developer would not need to make any code changes for model integration.

3.1. Introduction

Engineered systems are generally composed from interdependent components that are themselves composed of other sub-components. Because of this the representation and analysis of the detailed interactions within and between the components comprising these engineered systems is critical. However, holistic modeling of these systems is challenging (Arnold, 2013). This is in part due to the size and complexity of detailed systems models and in larger part because of the need to organize and integrate a collection of models representing the components and subcomponents of the system into a single coherent information artifact. Particularly because in many cases each of the component and subcomponent models are developed by separate teams of analysts (or individual analysts) with differing domain knowledge, differing modeling practices, and differing expectations as to the outcomes of the modeling and analysis process.

Integrated modeling (Laniak et al., 2013; Muth and Bryden, 2013) seeks to address these challenges and "includes a set of interdependent science-based components (models, data, and assessment methods) that together form the basis for constructing an appropriate modeling system" (Laniak et al., 2013). In a traditional modeling approach, the individual component models are linked with each other using software/code. In many scientific and engineering applications each of the submodels is incorporated as a subroutine within the larger systems code. In other cases each of the models is a software library that exposes application programming interfaces (APIs) (Michel, 2013). All the linking occurs in a single software program that brings together the individual models, including problem specific entities like

initial conditions, boundary conditions and information/data transfer between models. All of which must be mediated and controlled by the applications developer. More often than not, the monolithic software program also incorporates elements of the hardware that it is going to run on. For example, it may make assumptions about the availability of certain compute clusters, parallel programming models etc. Although this approach can be effective in solving the problem at hand, it makes it difficult for another engineer to reuse this work without undertaking the significant effort of refactoring the code to suit the new problem to be solved.

One way of overcoming the problem of building and modeling large-scale systems models is by taking an *integration framework* based approach, where each individual model is part of a larger framework of models. This use of the term *framework* is consistent with the definition used by Rizzoli et al. (2008) to describe the integration frameworks used in environmental modeling. That is "a set of software libraries, classes, and components, which can be (re-) used to assemble and deliver an environmental decision support system (EDSS) or an integrated assessment tool (IAT) to support modeling and processing of environmental knowledge and to enhance the re-usability and distribution of such knowledge." There are a wide range of open source integration frameworks (e.g., SCIRun (SCI, 2016), OpenDX (2011), Common Component Architecture (CCA)-capable CCaffeine (Allan, 2005; Bernholdt et al., 2006), Object Modeling System (OMS) (Lloyd et al., 2011; Ascough et al., 2005; David et al., 2002), The Invisible Modeling Environment (TIME) (Rahman et al., 2003), Open Modelling Interface (OpenMI) (Gregersen et al., 2007; Blind and Gregersen, 2005), and VE-Suite (Bryden and McCorkle, 2004) and closed-source integration packages (e.g., Matlab's Simulink™ (MathWorks, 2016), Execution Engine ™ (Simulia, 2016), ModelCenter™ (Phoenix Integration, 2016), and Protrax™ (2015). In general these integration frameworks are targeted at a specific

applications (e.g., SciRun is focused on computational steering and does not support integrating generic simulation and modeling tools, Protrax is used for modeling large processing plants, and OpenDX is for visualization integration), provide support for particular types of models (e.g., ModelCenter™, Execution Engine™, and Matlab's Simulink™ all provide support for the integration of specific sets of tools), or have specific model integration needs that limit the development of generic system models (e.g., TIME requires utilization of .NET as the development environment and OMS 3.0 requires access to source code for the models being integrated). In each of these instances integration of the code is based on enabling the coupling of the models together by linking the inputs and the outputs together in the manner of message passing between the component packages (i.e., models) in a way the requires adherence to a given data standard or requires user intervention to identify and manage the data flow.

The environmental modeling community, driven by the need to coordinate between various disciplines and to integrate the modeling efforts of these disparate groups, has been active in the development of a number of general purpose model integration tools (Laniak et al., 2013). These efforts include the development of the Bespoke Framework Generator (Armstrong, Ford and Riley, 2009), the Earth System Modeling Framework – Flexible Modeling System (Balaji, 2002), OASIS (Ocean Atmosphere Sea Ice Soil) (Redler, Valke, and Ritzdorf, 2010), and CSDMS (Community Surface Dynamics Modeling System) (Peckham, Hutton and Norris, 2013). All of these systems require models to have *initialize*, *run*, *finalize*, *get*, and *set* functions for basic control over the models; provide a code based method for connecting the models together, using some form of XML file or some other type of configuration file, to create a low-level model-to-model interface; and require an agreed upon global ontology describing the variables passed between models. OASIS and CSDMS also provide GUIs for connecting the

models into an interface for visualization and automated configuration file generation. All of the integration frameworks other than CSDMS require that the models in the system use the same programming language. CSDMS uses Babel (Babel 2016) to support models in C++, C, Fortran, XML, Python, and Java. While the approach taken by these packages is effective, it requires significant coordination and cooperation between the various modeling teams and it compiles a single executable. If a model is going to be integrated into another system same type of coordination between the various modeling teams in the new system is required.

This overall integration framework approach establishes bounds on model developers wherein each model has to satisfy a set of criteria to fit into the integrated modeling framework. An advantage of such an integrated framework is that all the individual models are consistent. Once a model developer understands the framework, developing newer models that are consistent with the established protocols of the framework becomes easier. On the other hand, developing a large scale systems model composed of separately developed models generally requires the development of a common (global) semantic schema and ontology, which can be a time consuming process. Additionally, imposing functional model development constraints on developers at a global level is a difficult task. The framework may not be sufficiently adaptable to incorporate some types of models, it may be difficult to incorporate legacy models and code into the integration framework, and some groups of developers may consider the global constraints too restrictive and need or want changes to the integrated framework. As a result, the integrated modeling framework changes with time and can cause version compatibility issues. Thus such a centralized approach can be useful for smaller groups of engineers, but can quickly become intractable for universal adoption.

In a departure from traditional engineering and scientific computing, the challenge of adapting and confirming to prescribed protocols can be overcome by taking the approach of developing loosely coupled and decentralized systems. In concept the decentralized system is very similar to a web-based application that, for example, predicts turnout at a public event based on location, weather and traffic conditions. Each of the public event listing, weather, and traffic conditions is an independent information service accessed over the Internet. Each of these services is developed independently of the others without making any assumptions or following common data interchange protocols. And the developers are free to define access, use, and content delivery protocols. And the users (the systems builders) are free to choose which information service to use. Only the service that combines the data from each of the three information services needs to know the protocol emitted by each service it is accessing. At a future date if the wrapper service needs to add, say, public transportation information, it can do so by calling yet another service that publishes this new piece of information. This can be done without renegotiating previously established protocols. Such a decentralized approach leads to a service-oriented architecture with cleaner interfaces and data transfer between each service (Perrey and Lycett, 2003).

In this article, a novel approach to developing a loosely coupled and decentralized integrated modeling environment appropriate for engineering and scientific computing is proposed, based on a federation of independent models each of which is an independent web-based model service (i.e., an information artifact) accessible via a web API using interaction protocols chosen by the model developer. In the proposed framework, this collection of independent models is joined together in a federation describing a particular system. In this way a complex system model can be built by composing together multiple individual models and can

be deployed as a system model. Furthermore, the developed models in the federation set do not have any schema imposed on the structure of inputs and outputs by the developer. The model developer decides the structure of inputs and outputs that the model accepts and emits, the only requirement being that this information be broadcast to model developers via API calls. In this way a component model can be used as-is in multiple systems models.

The communication between constituent models is orchestrated by a federation management system (FMS). The FMS is aware of all the models that are registered members of the federation and has the ability to trigger the execution of any of the registered models as needed. A user communicates with the FMS by initializing the desired systems model, describing a list of individual models that constitute it and the actions the models need to take, and supplying any additional information needed (e.g., boundary conditions, system constraints, design parameters). The FMS is then responsible for the communication between each constituent model. The FMS is also responsible for the availability of computational resources to execute any registered model by starting up new instances of a model based on usage.

For such a decentralized system of models to be functional and scalable a cloud-based architecture is the most practical solution. The primary reasons for this are

- Universal availability—Cloud platforms can be accessed by anyone with a web-browser and an Internet connection. This opens up the possibility for model developers to build and publish their models either as part of a closed group or globally regardless of their geographic location. Model developers should be able to publish their models by registering them with the FMS.
- Scalable platforms—As more models are added the computational resources can be increased automatically without human intervention. The deployed models need to be

readily available every time the FMS sends a request for computation. Cloud platforms are inherently scalable in terms of hardware and the process of scaling can be made intelligent and automated by utilizing platform level APIs from the providers. Thus, highly scalable and fault-tolerant software systems can be developed.

- Cost effective—Cloud platforms work on a cost per usage model and are hence cost effective because the user only pays for the compute time and not for procuring, provisioning and maintaining the compute, storage and networking resources.

3.2. Background

3.2.1. Cloud computing

Traditional HPC platforms have been built using custom hardware to provide massively scalable platforms for scientific computing applications. The number of floating-point operations per second delivered by this class of hardware is much greater than compute clusters built using commercial grade hardware. The pursuit of grand challenge scientific modeling and computational problems has warranted the purchase of expensive hardware for cutting-edge scientific research. However, this has made it difficult for engineering practitioners to solve engineering problems using HPC clusters because the hardware is deployed primarily towards solving the most difficult research problems. Thus the cost of HPC hardware can be prohibitive for most engineering problems in practice.

Over the last few years with the rise of cloud computing, it has become easier to obtain compute cycles on an on-demand basis (Amazon, 2016). Several companies have built data centers and technology platforms using commercial-off-the-shelf hardware and are making them available over the Internet. This has enabled applications to scale dynamically to support

millions of concurrent users for consumer applications. The same technology platforms are now being used in traditional enterprise applications blurring the difference between consumer and enterprise applications (Giessmann et al., 2012). Software vendors now host thousands of applications in the cloud that can be accessed by its users through a web-browser. Cloud computing eliminates the purchase of expensive processors, networking and storage resources by making them available over the Internet (Amazon, 2016). The availability of high-end hardware on an on-demand basis makes it possible to move many types of engineering, scientific and HPC applications and workflows into the cloud.

This has opened up new opportunities for creating novel scientific, data management, analysis and visualization applications that leverage the performance capabilities offered by cloud computing. Engineering modeling and scientific computing problems can now be solved leveraging cloud-computing capabilities. By harnessing the computing power of the cloud, smaller form-factor devices can also be integrated into engineering workflows and to perform operational tasks in the field that might not be currently possible. It also becomes possible to scale applications at a lower cost per processing/computing unit (Armbrust et al., 2010).

Consider the example of a cloud based traffic reporting application that helps drivers choose faster routes by making congestion data openly consumable. With ubiquitously available computing resources and publicly available data, the decision to change a route can be easily made either by a software system or a human being. Also, these decisions can be made and shared continuously in real-time independently by multiple parties leading to better co-ordination between people using the service. Furthermore, software applications can be written using this data in conjunction with other data sources to add more value to the existing service as well as creating other independent services as a result. Thus a variety of software applications get built

independently that gives different contexts to ever growing sources of data, services and algorithmic techniques.

A similar argument can be made for publishing and sharing engineering algorithms and information within an organization or globally. Availability of the algorithms, data, information and analysis opens up access to various people within an organization that can make decisions based on this data. This in turn can trigger enhanced collaboration within and even across organizations to help speed up engineering design, problem solving, and decision-making tasks. Also new applications and workflows can make various types of engineering data, analysis, models etc. available to people in a group or organization. In addition non-engineering data such as bill-of-materials, cost models and project timelines can also be tracked and managed using cloud computing resources. These new applications augment and enhance existing methodologies and workflows by making the data more consumable and in the process, opening up new insights.

Several researchers (Vöckler et al., 2011; Jorissen et al., 2012) as well as commercial entities (Amazon, 2016; Microsoft, 2016) have made attempts to run HPC workloads on cloud computing resources. Data from astronomical measurements was processed utilizing cloud computing resources from Amazon, Nimbus and Eucalyptus by provisioning cloud computing resources, mapping workloads to them and de-provisioning the resources on completion (Vöckler et al., 2011). They found that cloud computing can be a viable solution for several scientific computing problems. As part of their research they also concluded "*being able to add and remove resources at runtime outweighs the networking and system management overheads*". A platform for scientific computing was developed and used for simulations in materials science (Jorissen et al., 2012) using the Amazon Elastic Compute Cloud (Amazon, 2016) to develop an

Amazon Machine Image that was the primary underlying technology for their platform. The authors solved two problems in materials simulations that involved loose and tight coupling of codes. They found that although the EC2 platform is not efficient for the transfer of large amounts of data, it is competitive in achieving the speedups similar to that provided by Infiniband clusters. A big data platform for scientific workflows was developed and used to solve an image processing problem (Zhao et al., 2014). They also compared the efficacy of multiple cloud platforms for performance, price and ease of provisioning and management of compute resources.

A series of experiments were developed to evaluate the Amazon EC2 infrastructure as an alternative for many-task computing workloads (Iosup et al., 2011). Although the authors found that the overall performance of commercial cloud hardware is low compared to dedicated HPC resources, but underscore that fact that commercial clouds can fill the gap for temporary and instant need for compute resources. The authors also acknowledge that the performance evaluations are expected to change with time as commercial cloud vendors improve their offerings. Amazon EC2 is now offering HPC grade hardware for scientific applications (Amazon, 2016). HPC instances were found to be 8.5 times faster than the original general-purpose cloud computing instances (Fox, 2011). This shows a trend of general-purpose compute instances becoming more powerful over time as the cost to run them goes down. Fox also points out, "*Many scientific computing problems do not require supercomputer performance but would benefit greatly from modest parallelism"*. HPC clusters were extended using EC2 based cloud clusters by Belgacem and Chopard, 2015. They found that with a load-balancing strategy they were able to benefit from utilizing the cloud computing resources, as opposed to merely adding

more machines to an existing cluster. The authors used MPI based models that were coupled and executed in a distributed manner.

The researchers above primarily have used cloud computing instances to extend existing HPC hardware and augment the time-to-solution for their specific scientific problems. Commercial cloud hardware was found lacking in some instances (Vöckler et al., 2011), but several researchers have acknowledged the power of being able to quickly provision and utilize on-demand compute resources (Belgacem and Chopard, 2015). Also, most of the computational workloads have been using traditional models like MPI, but running on cloud hardware. So most of the problems solved in the literature are related to comparing performance of existing codes running in the cloud. Tightly coupled models have also been studied several of which have used MPI or other software libraries to distribute and combine tasks and results over distributed computing resources.

3.2.2. Stateless Models

The concept of state is critical to the implementation of federated model sets described here. State refers to the entirety of information that defines a model while executing a computational task. For example, consider the solution of an ordinary differential equation using a fourth order Runge-Kutta (RK4) method. The initial time-step, initial conditions, constants in the evolution equation and the current time-step define the state of the RK4 model. If the state within a model is continued to be maintained beyond the execution time-frame, the state is said to persist. This is typically the case in monolithic computer codes used in systems models where state in one model is continued to be maintained while a different model is executing based on the state from the first model.

However, if the model does not retain any state between invocations it is said to be stateless (Thönes, 2015). If it is possible for the model to pass-on its state information to the next model without persistence via shared-memory or some message passing mechanism, the individual models themselves become reusable computational entities across systems and multiple system models. Any system model can use an individual component model as needed. As noted above, this work requires that each individual model is stateless i.e., the model does not persist state information beyond the executing time-frame for the current task. This is an important consideration because it enables each model to act as a "functional unit" that can be reused and combined easily with other models. It must be noted that stateless models can be created either as a single solver, say, an RK4 solver or it could be a combination of multiple solvers. In stateless models the output is dependent only on the input.

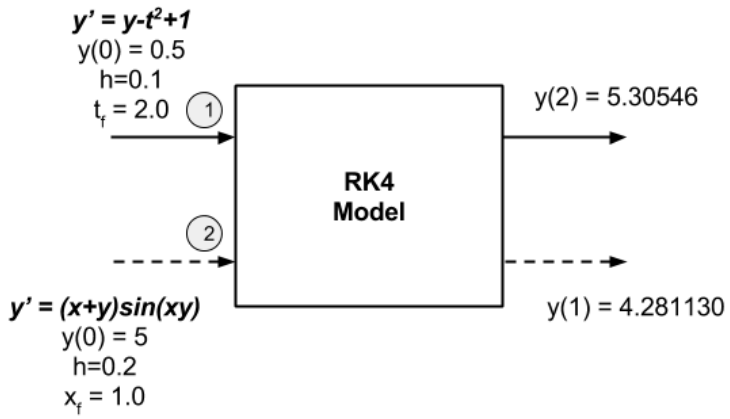

Figure 3.1. Stateless model that implements Runge-Kutta 4th order integration.

There is no additional information needed i.e., given the same inputs the model returns the same output each time. Participation within a federation of independent models requires that

each model be stateless and that each model identify its inputs and outputs. An example of a simple stateless RK4 model is shown in Figure 3.1. The evolution equation, initial condition, the step-size and the final value are shown as inputs to the model on the left. Correspondingly, the output from the model is shown on the right. Two different sets of such inputs and outputs are shown in the figure, each of which is invoked at different times, possibly by different users. User 1 invokes the RK4 model and once the service accepts this request it does not accept any further requests. All the information entering the model (on the left) is the state information that the service uses to evaluate the solution for $t=2.0$. Thus, all intermediate states are tracked by the service and these states are used for completing the computation. However, once the final step is reached, the service releases all state information and sends a response back to user 1. On completion of this computation, the service it now ready to accept and compute the request from user 2, which proceeds in a manner similar to the above description. If user 1 would like to further integrate the equation until $t=4.0$, this can be sent as another request to the service using the response from the previous integration. Thus, the service depicted is an example of a stateless service that keeps state only for the duration of a request and then discards this information on processing the request and responding to the user.

3.2.3. Microservices Architecture

To integrate software written and maintained by disparate teams, the popular choices are, service oriented architectures (SOA) and microservices architectures (Erl, 2005; Thönes, 2015). A similarity between each of these architectures is their focus on development and deployment of modular services i.e., services that perform specific functions. The major difference between the two lies in the functionality that is encompassed by the modular services. In SOA, the modular components tend to be functionally similar and thus tend to be large and abstractions of

multiple code bases that perform a similar function. On the other hand, in microservices based architectures, the delineation between microservices is based on the logic incorporated in the service as opposed to functionality. The advantage of this approach is that each microservice is a stand-alone service with a well-defined interface, using which the microservice can be invoked by other microservices or users. More information on these architectures can be found in Villamizar, 2015 and Thönes, 2015.

The approach taken in this article is towards developing engineering models as mircoservices. Microservices are independent software services that are designed to perform a focused function (Thönes, 2015). In general, microservices meet certain requirements:

- They must be independently deployable and managed
- They must publish their inputs and outputs
- They must have the ability to be scalable on cloud-platforms

Microservices can thus be developed by independent developers, deployed, and consumed by anyone needing that service in an easy manner, typically HTTP endpoints. In the context of this article, microservices are viewed as services that provide a solution to an engineering problem by performing a computation. For example, a 2D-Poisson grid-generation code can be a microservice that takes the inputs of the domain boundary and returns the computed grid in a certain format. Since engineering models are tightly tied to the initial state and boundary conditions, an additional requirement for the engineering modeling microservices that is proposed in this research is that the microservices must be stateless, as described in the section 2.2. This ensures that the model can be invoked by any other service that needs this computation as long as the invoking service can provide the inputs in the prescribed format. In addition, from the perspective of a cloud-based applications, stateless microservice architectures

are easier to scale up or down based on usage and demand requirements (Thönes, 2015). Since microservices are built on the premise of accomplishing a single functional or computational task, scaling them involves scaling only those services that are under a heavy load. As an extension, since only a targeted subset of services get scaled, the approach taken is more streamlined from scaling SOA based services that can involve coupling (Thönes, 2015; Erl, 2005).

3.3. Problem Description

In this work, a novel methodology is proposed where cloud computing is used as the primary compute infrastructure for engineering modeling and computations, instead of augmenting existing hardware resources. Individual engineering models are enabled to be scheduled and execute on-demand, and more complex models built from foundational models. This approach abstracts out the details of the algorithms from the data passing and scheduling mechanisms, which enables the foundational models, and algorithms to be reused easily independent of the context of the problem being solved. Based on this the goal is to have the ability to build systems of models from simpler models and to build the software infrastructure to enable this. These models can include engineering models like finite element analysis and computational fluid dynamics as well as non-engineering models including cost models and product diffusion models. In this work we propose an approach to build models as compositions of individual computations that form a linked system of models that solves a larger problem. A primary driver for this approach is the advent of cloud computing which enables practically infinite scalability at a relatively low price point compared to traditional HPC systems. Existing cloud computing platforms abstract away from the user details of processors, storage and networking by making them available on demand. It is now possible to build new services

leveraging the maturity of these platforms. The authors envision such a platform approach to linking models as “Modeling-as-a-Service” where users can access and build complicated models based on sets of foundational models available for use over the Internet. In addition, users can add their own custom models by registering them with the FMS. This requires

1. A set of models that solve specific problems. These models can be self-contained individual solvers that implement a specific algorithm or can be composed from other existing models. Users must have the ability to add more models to the federation as they see fit.
2. A management system that can accept new models and broadcast the details about existing models in the federation set. This federation management system (FMS) needs to be able to accept requests from users and orchestrate, coordinate and execute models in the prescribed order necessary to solve the problem.
3. A framework of communication between models and the FMS where each model can receive instructions to execute and notify the FMS of the completion of a computation or of errors.

This paper proposes a novel methodology to compose complex systems from simpler ones using a distributed systems approach where individual services are orchestrated by the FMS. Using the proposed FMS a user has the ability to compose together multiple individual computational models to develop a systems model of a more complex system. In the process, the FMS checks for feasibility of interaction between the computational models and proceeds with composition only if the appropriate feasibility tests are valid. In this work, the ability to evaluate engineering models leveraging existing code and frameworks is also incorporated. This allows for seamless utilization of existing code bases.

However, solving complex problems in a decentralized manner is a challenging task due to:

1. Coordination of the model services. Since multiple services will be utilized during the execution of a complex model they need to be executed in the correct order to obtain accurate results. In this work, the FMS is responsible for all coordination activities between models in the federated set.
2. Exchange of state information between models at runtime. Since each model after execution does not retain any state information, the state at the end of an execution must be retained and passed onto the next service in the execution workflow. To accomplish this, a queuing system is used to pass messages between multiple distinct solver sub-systems/models. The queuing system coupled with the messaging schema is the primary communication mechanism between the solvers and the FMS.

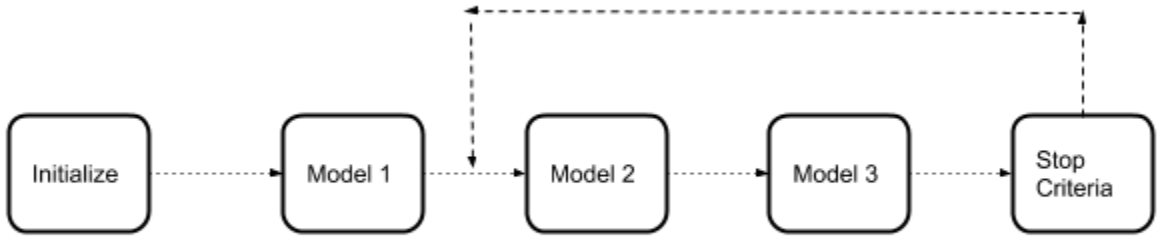

Figure 3.2. An example of a task workflow.

3.4. Methodology

As discussed in Section 3.3 the federation management needs to perform the following specific tasks

- Allow users to register new models
- Broadcast information about available models
- Accept user requests for performing computations
- Schedule user requests and execute appropriate models

If a set of models are linked and invoked by the user to perform a single computation, the FMS should be capable of executing all of them in the prescribed sequence without delay. Additionally, the framework should allow the models to access a common file system where each solver can store and access intermediate data that is exchanged between the solvers. Building an infrastructure to support this requires several key pieces. These are

1. Queuing Service
2. Federation Management System
3. Solver Sub-system
4. Namespaces
5. Data Access Layer
6. Caching and Storage Layers
7. Web Application Programming Interfaces

Each of these is further described in the remainder of this section.

3.4.1. A Solver Sub-system Approach

In this work, the authors take a solver sub-system approach to developing a loosely coupled distributed system. Each solver is an individual sub-system of the overall federated system of solvers. Each solver sub-system constitutes of an algorithm or can be the composition of other solvers. For example, a Runge-Kutta 4^{th} order (RK4) integration algorithm is an individual solver that accepts the input evolution functions and initial state in an input format, solves the equations for a prescribed number of time-steps and returns the results (state) in an output format. This set of transactions is a compute job. After performing a compute job the RK4 solver does not hold or persist state for any further compute jobs that may be coupled to the previous task. The state is instead encapsulated in a message that is sent back to a federation management system (FMS) that schedules a subsequent job with this result. In the event that the compute job is complete (convergence or evolution to prescribed time-steps) the completion information is sent to a task completion queue for delivery to the user that initiated the sequence of jobs. Since a solver subsystem does not persist state locally, it is free to pick up another compute job as the situation demands. A manager-worker architecture is not imposed and hence there are no single points of failure, the advantage being that it becomes possible to scale the model microservices with relative ease. The approach decouples the scheduler, the queuing/messaging service and data from the solvers, so they can operate as individual functional units to perform operations as triggered by the messages they receive.

3.4.1.1. Queuing Service/Layer

The queuing layer ties together individual subsystem components in a loosely coupled manner. The queuing service forms the foundation for the channels that individual models use to communicate with the federation management system. Messages sent from models

to the FMS and vice-versa are all sent via the channels through the queuing system. Messages are processed by sub-systems in the order they are received, since the underlying queuing service satisfies first-in-first-out (FIFO) characteristics. A channel is an abstraction layer over the queuing system that defines where a sub-system sends or receives a message. Each channel is tied to every sub-system to either send or receive messages. A generic message structure is published by the FMS and the model services derive from this structure and extend it by defining their own message formats.

3.4.1.2. Federation Management System

The federation management system (FMS) is the primary orchestrator of the various solver subsystems. It has two primary responsibilities a) accept user requests and b) schedule and orchestrate/coordinate the appropriate solvers needed to solve the workflow at hand. To meet these requirements the FMS has two ingress channels one to accept user requests and one to receive messages from registered services, as shown in Figure 3.3. The channel for user requests, as the name suggests, only accepts requests from a user. The second ingress channel, the job scheduler channel, is used by models to communicate with the FMS. Each of these input channels has a thread pool that listens for messages coming through; a user request (UR) thread pool and a job scheduler (JS) thread pool.

A thread in the UR thread pool accepts a task workflow from the user and checks if the requested models have been registered. In the event that any of the models required to complete the workflow have never been registered, an appropriate error message is returned immediately and the workflow is rejected. If all the required models have been registered, the first task in the workflow is scheduled to run. This is the only time a thread in the UR pool schedules a task. Models never communicate with the FMS on the user request channel. Once

this first task has been completed, the model notifies the job-scheduler channel in the FMS with the completion status and current state information. The FMS then schedules the next task in the workflow to the appropriate model, which again notifies the FMS on its completion. This process continues until convergence has been reached or the requested number of time-steps in the have been reached.

Every model has to register with the FMS in order for the FMS to know about its availability. In the process of registering with the FMS the model has to advertise the computation that it can perform and the channel that the model is listening on. The negotiations that occur between a model and the FMS are shown in Figure 3.4. As soon as the model executable starts up, it sends a model identification, its message contract information and the name of its ingress channel to the FMS. The FMS checks this information in its database of previously registered model microservice if this information is valid and that there are no conflicts in either the model identification or the ingress channel names. This is an important step in order to ensure that the appropriate compute information is assigned the appropriate task. Once verified the FMS sends back an appropriate response to the model service and it is registered. In case any conflicts were detected, the FMS sends a response with an appropriate error message and the model service logs the message. The developer of the model in this case would have to make changes to the solver ID and ingress channel names, as appropriate.

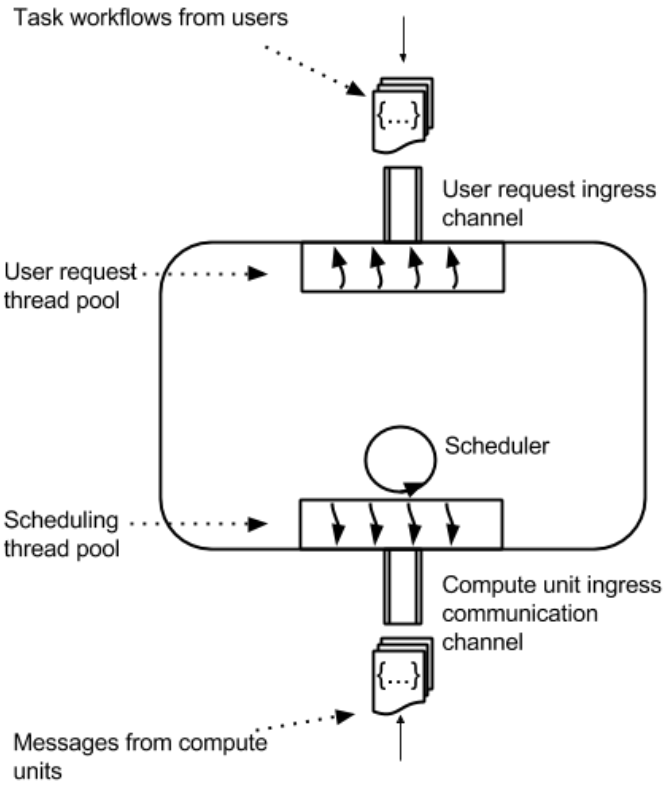

Figure 3.3. Federation management system.

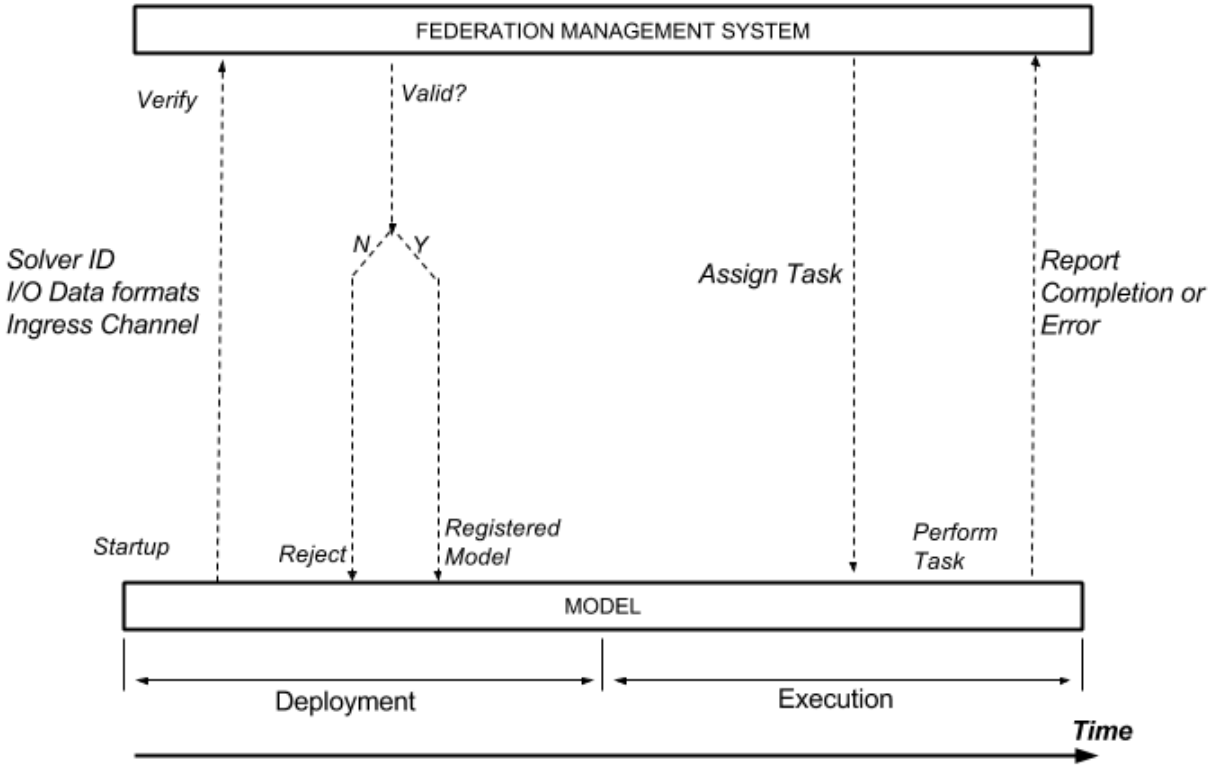

Figure 3.4. Timeline of interactions between a model and the FMS.

3.4.1.3. Data Access Layer

The data access layer is a software layer over the caching and cloud storage layers. It is used primarily by the namespace manager to store retrieve data from the namespace corresponding to a task workflow.

3.4.1.4. Message Contracts

This section describes the message contracts that need to be established between a model and the FMS. Since the architecture developed in this work is loosely coupled, information exchange between models and the FMS is critical to the functioning of the entire system. Furthermore, since the FMS orchestrates and schedules various models in the appropriate sequence to drive the workflow to convergence, the information exchanged between them must reflect the state of the problem being solved adequately. This aspect is also critical because the models are stateless.

Information is exchanged between various subsystems in the form of messages in the JavaScript Object Notation (JSON) (Nolan, 2014). This structure was chosen due to its simplicity and interoperability with multiple languages. JSON is primarily a text based key-value pair data structure, which make it easy to construct and read. This property is useful to developers and engineers for creating message formats and debugging them. Most popular languages in use today either have built-in capabilities or have mature libraries that can be included to parse, read and write JSON objects. This helps maintain programming language independence amongst the models and the FMS. JSON is also an accepted standard for various Internet based APIs lately. This section explains in detail the design of message structures used in this work.

For any two independently developed codes to exchange information, a "contract" has to be established between them to do so efficiently. This is relatively simple in tightly coupled software systems where these contracts can be re-negotiated and all the individual models can be updated as needed. However, in loosely coupled software systems this is challenging because some parts of the system cannot be changed to accommodate changes in other models that may have been developed by multiple independent developers. Hence changing something fundamental to the integrity of the entire system can be impractical.

Every message passed between a model and the FMS has two parts to the contract. One part of the contract is utilized by only the FMS to make decisions about scheduling and orchestrating models. The second part of the contract is used by the solver to load its initial state, read solver parameter information and the output location in the namespace used to write intermediate data.

As a simple illustrative example a system model is considered with an addition model and a multiplication model as constitutive models. They are used by the system builder to find the result of the operation $(a+b)*c$. The addition is performed first by passing in the values of a and b. The resulting value and a value of c are then passed to the multiplication model. Once the second computation is complete the final result is passed back to the user. Figure (3.5) shows an example of such a message structure exchanged between solvers and the FMS. The list of solvers

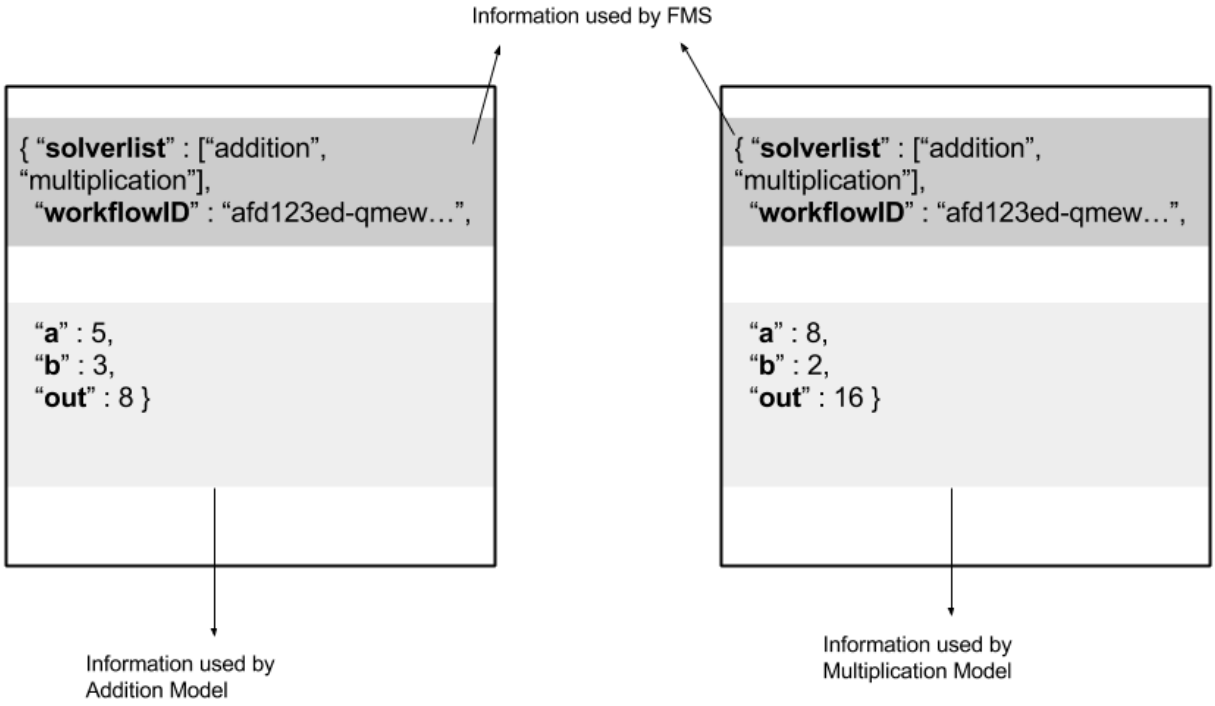

Figure 3.5. Example of a message contract.

that need to be utilized in the system model is specified by the system builder, while the workflow ID is assigned by the FMS.

The message contract can include any information deemed necessary by the model developer. The only requirement is that the basic information needed by the FMS be included in the message. For this purpose and for easier programmability, a model specific message is derived from a generic message class that enforces the necessary contracts. The model must then implement those contracts for it to be registered with the FMS.

This approach makes the FMS generic enough where it can function with any model. Also, the solvers and logic within the models can be developed independently without enforcing language or programmability restrictions. The message contract structure is critical to the functioning of the entire system in a loosely coupled manner. Further details about how each solver change message parameters during the execution of a workflow are explained in the examples.

3.4.1.5. Namespaces

As engineers create computational models, they are added to a database and assigned to an appropriate namespace. The namespaces are necessary to partition data in a logical and intuitive manner and enables sharing of the data between multiple users or software systems. As changes to the computational models are made, they are “checked-in” to the corresponding namespace in the database and updated accordingly. The metadata associated with each set of models is also updated as needed. The metadata may include information such as the geometry, material properties, grid information etc. This is an important aspect, because it reduces the time to extract useful information about a set of models. Users and automated software applications can thus use the metadata to obtain information about the namespace, instead of going through

each computational model. The metadata reflects a higher-level abstraction of information pertaining to the computational models in the namespace. It should be noted that the process of adding and updating models is a continuous process during the entire lifecycle of the design. Each of these instances can function independently or in concordance with one or more of the other instances in the collective to help solve a more complex problem.

3.4.1.6. Web Application Programming Interfaces

Web APIs are used to hide the server-side complexity by incorporating processing logic and exposing only the limited set of allowable inputs as HTTP endpoints for the system model to specify. In this way system models can access engineering model computations and metadata without knowledge of the details of the interactions, but only of the exposed interfaces of the Web API. In this work RESTful web APIs (Fielding, 2002) were developed to expose HTTP endpoints of the individual models. Using the exposed APIs a system builder can assemble multiple models into a single model space which can in-turn be exposed as another HTTP endpoint for reuse by other systems. Thus web APIs are critical in hiding complexity of the existing models while enabling their efficient reuse. Some examples of the Web APIs are described in Table 3.1 below.

Table 3.1. Examples of API endpoints and their functionality.

API endpoint	Description	Example

/fms/post	A POST request to the FMS with a payload that contains model(s) to be used and their inputs.
	The response from this request is an ID that can be used to track progress.
		/fms/post
		PAYLOAD :
		{
		“solverlist”:[”addition,multiplication”],
		“a”:5,
		“b”:3
		}

/fms/getinfo?model=<modelname>	A GET request to retrieve input payload and output format information about the specified model.	/fms/getinfo?model=”addition” Gets input and output information about the addition model.
/fms/getmodels	A GET request to retrieve all the models registered with the FMS.	/fms/getmodels Returns a list of registered models for the system builder to utilize.

3.4.1.7. Model Software Development Kit (SDK)

As described above, each model interacts with the FMS by registering with the FMS, reading messages from the FMS and sending messages back to the FMS. To encapsulate these “boiler-plate” interactions an SDK has been developed that enforces a programmatic contract between the individual models and the FMS. The SDK creates an interface between the FMS and the models in such a way that the FMS does not need to know the details of model computations while the model has all the necessary information to communicate with the FMS. The SDK also provides generic message contract classes for the models to inherit, which enables model developers to implement their own custom messages without disrupting the necessary message construct needed by the FMS. As shown in Figure 3.6, before performing any computations the models load their initial state after receiving a message from the FMS. When the computation is complete, the model sends the message back to the FMS and discards all state information. This

ensure that the stateless condition of the individual model and prepares it for use another system model. In addition, the Model SDK also incorporates persistence and caching objects for use by model developers. This allows model developer in focusing on the computational logic primarily and making use of the readily available persistence and caching mechanisms seamlessly. This approach also enables uniformity at the infrastructure level in order for models to be reused.

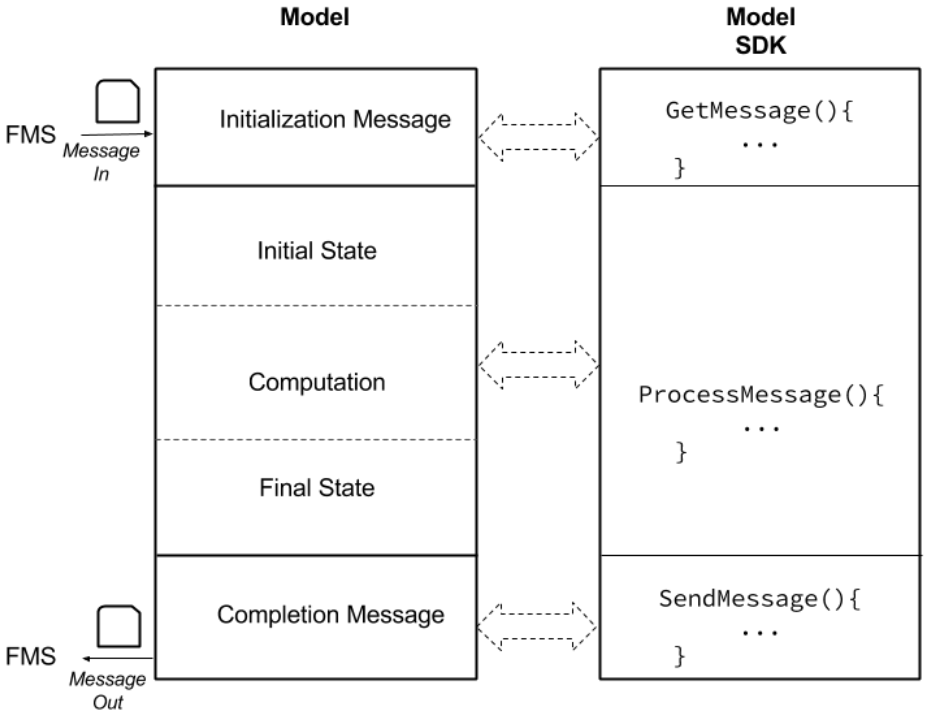

Figure 3.6. Representation of the Model and the Model SDK.

3.5. Architecture

The overall architecture of such a system is shown in Figure 3.7. In traditional approaches to linking engineering models the “linkage” occurs in code i.e. information transfer between models has to be incorporated in code. Thus the model-associated state information is tightly coupled to the models being used and to the execution context of the engineering problem. This makes the coupled code difficult to reuse without modifications and refactoring because of the state information is embedded into this coupled model. A primary reason for this is the availability of models as software libraries as opposed to individual atomic compute “engines” which act on request to a specified set of boundary conditions. Such an approach would require a loosely coupled distributed system based approach to building each model, scheduling their execution and orchestrating information transfer. An important requirement is that state information be moved away from individual models as they run. This enables any model of the same type can carry out subsequent runs without keeping track of state information. This is an important requirement when linking models together, because information transfer from one model to another can occur by models loading the updated state at run-time. This is an important distinction that the authors would like to note, from a “library approach” where this coupling occurs primarily at compile time of the code. This also plays a key role towards enabling a fault-tolerant and scalable distributed system to be constructed of collections of engineering models. Moving the contextual state information away from a model helps make it a foundational construct within a larger collection, which can be called to perform a task. Existing cloud computing platforms can be used to “spin-up” new model instances as per usage and demand requirements.

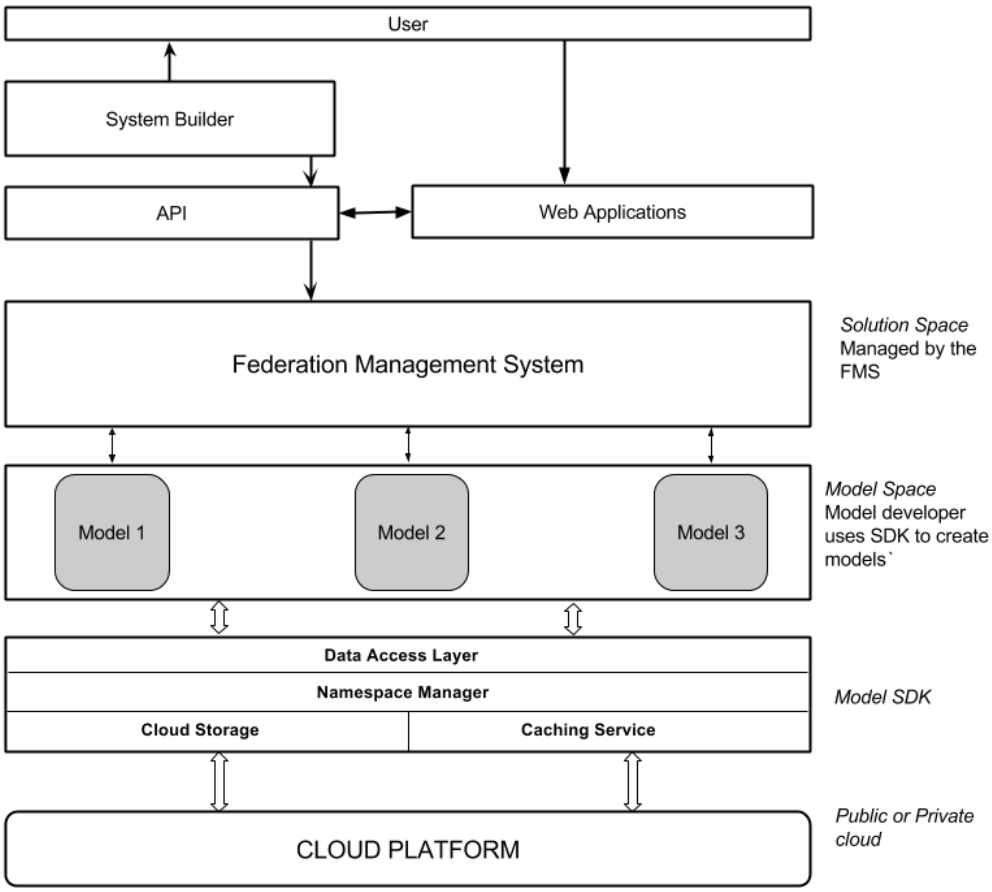

Figure 3.7. Architecture of the distributed system to compose computational models.

For this architecture to be functional it is important to address three key elements.

3.5.1. Information exchange protocol

For traditional system models to exchange information successfully between multiple models requires the development of protocols that must be adhered to by all models. However, for federated model systems i.e., loosely coupled systems it becomes difficult to develop and maintain standardized information protocols. For this reason, in this work each individual model publishes its own information exchange protocol for publishing and receiving data and analysis. This is similar to web application programming interfaces (APIs) (Michel, 2013). Using API based protocols gives each component model control and independence over the data it can receive and publish and eliminates errors due to incompatible protocols, leading to a service-oriented architecture (Erl, 2005). This is advantageous because the component models can be developed independent of each other. For this purpose, message contracts are developed for all transactions between component models and the FMS.

3.5.2. Information routing

The composition of the individual sub-systems is of critical importance as it affects the validity of the final result of the engineering problem. In this work the user determines the order of composition of sub-systems, using which a directed graph is generated. The directed graph specifies the execution order of individual subsystems (McNunn and Bryden, 2013). This is a critical step since some subsystems may have to exchange information iteratively to converge. Once the order of is determined, it is submitted to the FMS, which schedules execution of the appropriate components.

3.5.3. Information compatibility

The metadata for an engineering model includes data such as the topology, operating range of the component, material characteristics, etc. This metadata is used to make decisions regarding the compatibility of connected domains. There are two primary issues with information compatibility:

a. Adjacency: Two adjacent models must always be compatible with one another with regard to their topologies, time and spatial domains. In addition, the types of information being exchanged must be the same. For example, information about temperature from one domain must be passed onto an appropriate temperature field in the second domain. Units and descriptors of the information being passed must also be the same and translators could be used to convert standard units as required.
b. Appropriate use: If two models satisfy the requirements of being compatible, it however, does not imply that they can be used adjacent to each other. There might be differences in material properties or operating ranges that the model might be specifically addressing. These engineering decisions need to be taken into account by the system builder during the process of assembling the system model.

It is important to understand if the sub-problems being considered can interact with one another. Although it is not necessary that all the domains should have feasible interactions, it is expected that the strongly connected components should respect this criterion.

3.6. Example Application: Cookstove Preliminary Design

MacCarty and Bryden have developed a steady-state heat transfer model for the conceptual design of a biomass cookstove (MacCarty and Bryden, in review). These low-cost technologies have been identified as an important option to help alleviate the impacts of energy

poverty in developing countries where over 2.4 billion people rely on open combustion of biomass to meet as much as 97% of their daily energy needs for cooking, heating, and lighting (IEA, 2010; Johnson and Bryden, 2012). The household air pollution associated with this inefficient and incomplete combustion has been attributed to an estimated 4 million premature deaths each year, representing the 2nd leading cause of death for women globally, and contributes to global climate change particularly due to emissions of black carbon particles (Lim et al., 2012; Bond et al., 2013). In these diverse communities, the cooking practices, available resources, and cultural preferences vary considerably on a local basis. Therefore, cookstove designs must be adapted and the use of modeling simulations in which the various design parameters are tailored to these communities can help to increase the efficiency of the design process.

A basic improved cookstove typically consists of 1) a combustion chamber, in some cases insulated, to enclose and shield the fire in order to contain the heat and generate more complete combustion; 2) a grate to elevate the fuel and allow better flow of air through the fuel bed; and 3) a flow path including channels to provide improved heat transfer to the bottom and in some cases the sides (when a pot shield is used) of the cooking pot. The stove is typically fired with wood of varying moisture content ranging in size from small twigs to large unsplit branches, although in some cases crop residues and dung may be used. The goal for designers is to develop designs that generate efficient heating and low emissions while still providing a user-friendly device that can operate at varying firepower, with a variety of cooking pots, and a range of fuels (MacCarty and Bryden, 2015).

The model for predicting the efficiency of heat transfer into the cooking pot was developed by breaking the system into three separate but coupled zones, including a) the fuel bed zone; b) the flame zone; and c) the convective heat transfer zone; which are in turn coupled to a

model of the air flow due to buoyancy and friction (Figure 3.8A). In the packed bed, solid phase combustion includes heating of the wood and drying of the fuel moisture followed by pyrolysis and char burning with primary air. In the flame zone, secondary air enters, is heated, and is supplied to gas phase combustion. In the heat transfer zone, energy is lost through the stove walls, transferred to the pot via convection and radiation, and exits as sensible losses. Fluid flow and the entrainment of excess air is driven by natural buoyancy, and is slowed by pressure losses due to friction throughout the various geometries of the flow path. The temperature and velocity profiles throughout the system are determined using traditional heat transfer and fluid flow theory as a function of fifteen design variables, including 10 geometric parameters (Figure 3.8B), 2 material properties, and three operating conditions (MacCarty and Bryden, in review).

In this incarnation, the model operates with initial estimations for flow and temperature profiles, and then progresses through the three zones up through the exit of the stove, at which point the velocity is evaluated. With this new flow rate, the model iterates through the zones again until convergence is reached. In this sense, the model is a singular entity that executes each of its three constitutive models within a monolithic piece of code. Most scientific codes are designed in this manner and do not permit model reuse and substitution with ease.

In the context of FMS, this method is extended by separating the individual models and re-constituting them using application programming interfaces (APIs) that are accessible over the Internet. Thus, there are individual services that execute the algorithms for each of the three models i.e. a bed model, a flame model and a heat transfer model. Each of these services is registered with the FMS and advertises the input data formats they accept and the output that is emitted.

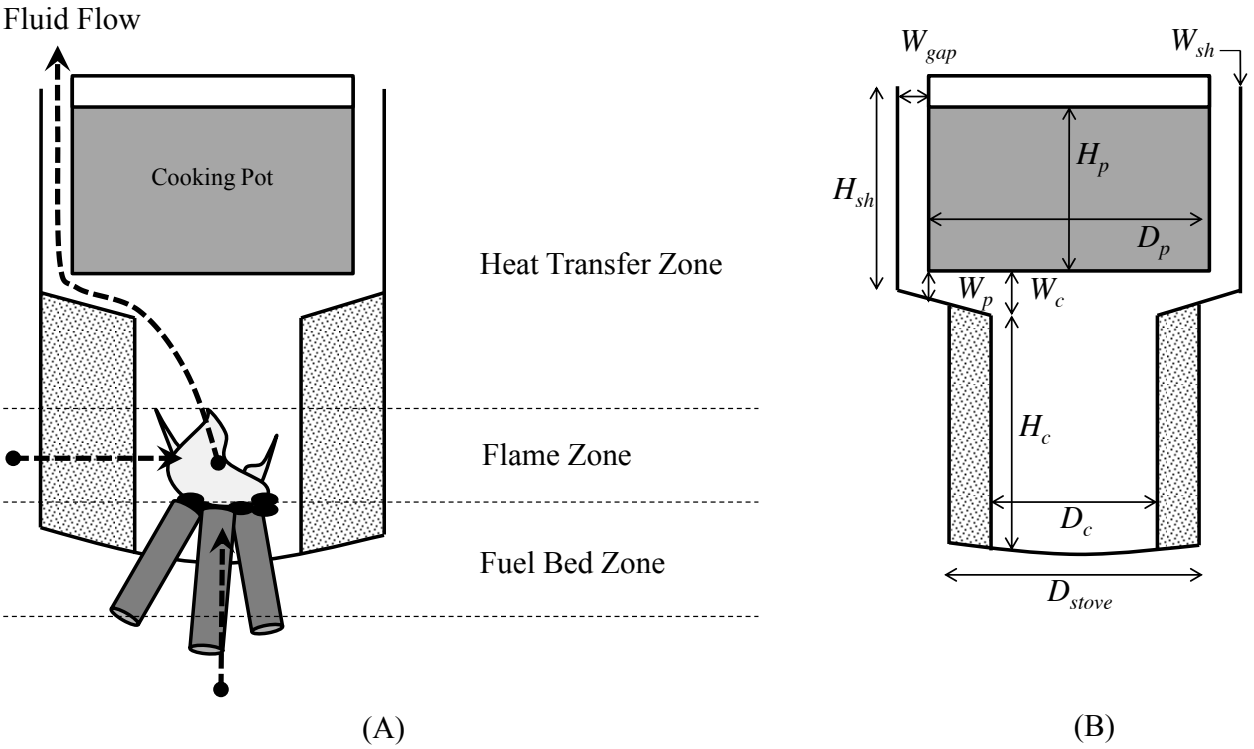

Figure 3.8. (A) The coupled zonal models of the cookstove system; (B) the geometrical design variables (after MacCarty and Bryden, 2015; MacCarty and Bryden, in review)

This allows each of the services to be called independently, if required. In addition to the individual services, a wrapper program has been written that encapsulates the sequence of API calls to each of the services in a modular fashion. The inputs and outputs to each model and the equations they solve are shown in Table 3.2. Thus if changes are made to the upstream constituent models, only the wrapper code needs to change. The configuration of services and the wrapper program with the FMS is shown in Figure 3.9.

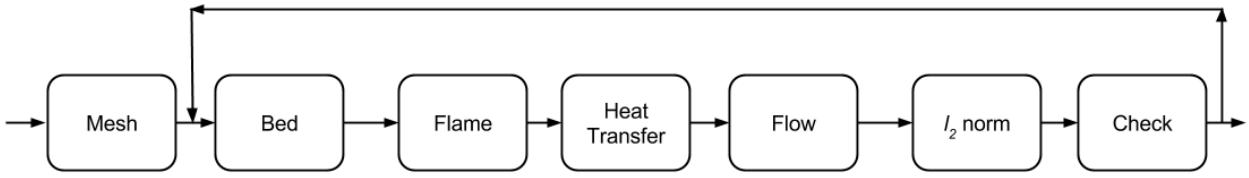

Figure 3.9. Flow of component models within the federated system of models.

3.7. Discussion and Results

The original software for the stove model was a single piece of code that solved all the above mentioned regions and finally reported efficiency. This model code was split into 6 individual component models chosen on the basis of the heat-transfer zones as shown in Figure 3.8 and the physics of the problem (MacCarty and Bryden, 2015). Table 3.2 shows the individual model equations with their inputs and outputs.

Table 3.2. Component models with their inputs and outputs.

<table>
<tr><td colspan="2">MODEL 1 – Mesh

Initializes the geometry and allocates variables for computations.</td></tr>
<tr><td>H_c, H_p, r_c, r_p</td><td>$\boldsymbol{A}_{flow}, \boldsymbol{f}, \boldsymbol{AHT}_{pot}, \boldsymbol{D}_h,$
$\boldsymbol{AHT}_{loss}, \boldsymbol{F}_{firebed-pot}, \boldsymbol{A}_{pot-bottom},$
$\boldsymbol{A}_{fuelbed-bottom}, A_{pot}, A_{wall}, A_{shield}, \boldsymbol{T}$</td></tr>
<tr><td colspan="2">MODEL 2 – Bed

Calculates rate of burning and production of fuel moisture and products of char combustion based on firepower, and applies conservation of energy to the char content of the fuel to determine the temperature of the fuelbed and exit gases.

Equations:
$\dot{m}_{char} HHV_{char} = \sum_i \dot{m}_i \left(h_{bed,i} - h_{amb,i}\right) + \dot{m}_w h_{fg} + \emptyset_{flame} \varepsilon_{char} \sigma A_{bed} F_{bed-pot} \left(T_{bed}^4 - T_{pot}^4\right) +$
$\emptyset_g \varepsilon_{char} \sigma A_{bed} (1 - F_{bed-pot})(T_{bed}^4 - T_{wall}^4)$</td></tr>
<tr><td>$LHV_{wood}, MC_{wood}, firepower,$ constants, T_{boil}, T_{amb}
h_{fg-H_2O}, y_{char}</td><td>$\boldsymbol{T}_{fuelbed}, \boldsymbol{T_q}, \dot{m}_{gv}, \dot{m}_{H_2O-mc}$</td></tr>
</table>

Table 3.2. (contd.). Component models with their inputs and outputs.

MODEL 3 – Flame

Determines rate of burning volatiles, and applies conservation of energy to the char content of the fuel to determine the temperature of gases exiting the flame zone.

Equations:

$$\dot{m}_v HHV_v + \left[\sum_j [\dot{m}_j h_j]_{bed} + [\dot{m}_{air2} h_{air2}]_{amb}\right]_{in} = \sum_j [\dot{m}_j h_j]_{out}$$

$$\dot{m}_{gas} = \dot{m}_{char} + \dot{m}_v + \dot{m}_w + \dot{m}_{air2}$$

$\boldsymbol{T}_{fuelbed}, \boldsymbol{T}_g, \dot{m}_{\boldsymbol{gv}}, \boldsymbol{T}_{amb}, \dot{m}_{gchar}, \dot{m}_{air}, q_{rad-flame},$ $\dot{m}_v, HHV_v, h_{fg-H_2O}, \dot{m}_{H_2O-mc}$	$\boldsymbol{T}_{flame}, \boldsymbol{T}, \dot{m}_v$

MODEL 4 – Heat Transfer

Applies conservation of energy to determine the heat transfer coefficients and flux, and resulting exit temperature through each differential control volume along the flow path.

Equations:

$$\sum_k \dot{m}_k\left(h_{flame,k} - h_{out,k}\right) = \emptyset_g \sigma A_{pot,rad} F_{bed-wall}\left(T_{int}^4 - T_{pot}^4\right) + \emptyset_{flame} \varepsilon_{char} \sigma A_{bed} F_{bed-wall}\left(T_{int}^4 - T_{bed}^4\right) + q_{wall}$$

$$q_{wall} = \frac{T_{in} - T_{amb}}{R_{int} + R_{cond} + R_{ext}} = \frac{T_{in} - T_{int}}{R_{int}} = \frac{T_{ext} - T_{amb}}{R_{ext}}$$

Table 3.2. (contd.). Component models with their inputs and outputs.

<table>
<tr><td colspan="2">Pot bottom center:
$$\sum_k \dot{m}_k \left(h_{in,k} - h_{out,k}\right) = \tilde{h}\pi \left(\frac{D_c}{2}\right)^2 \left(T_{in} - T_{pot}\right)$$
Pot bottom above stove:
$$\sum_k \dot{m}_k \left(h_{in,k} - h_{out,k}\right) = \tilde{h}A_{pot}\left(T_{in} - T_{pot}\right) + q_{wall} + \emptyset_g \sigma A_{pot}\left(T_{int}^4 - T_{pot}^4\right)$$
Pot corner:
$$\sum_k \dot{m}_k \left(h_{in,k} - h_{out,k}\right) = q_{wall,stovetop} + q_{wall,shield}$$</td></tr>
<tr><td>$\boldsymbol{T}_{fuelbed}, \boldsymbol{T}_{flame}, \boldsymbol{T}, \dot{\boldsymbol{m}}_{\boldsymbol{gas}}, \dot{m}_{H_2O-mc}, \boldsymbol{A}_{\boldsymbol{flow}}, \boldsymbol{A}_{\boldsymbol{wall}},$
$\boldsymbol{A}_{fuel-bed}, \emptyset_{flame}, \emptyset_{gas}, \boldsymbol{T}_{pot}, \boldsymbol{T}_{firebed-pot}, \boldsymbol{\varepsilon}_{char},$
$\boldsymbol{T}_{int}, \boldsymbol{T}_{ext} \boldsymbol{T}_{wall}$</td><td>$\boldsymbol{T}, q_{pot}, q_{loss-wall}, q_{rad-pot}, q_{conv-pot}, eff$</td></tr>
<tr><td colspan="2">MODEL 5 – Flow
Determines the velocity and pressure drop as a function of temperature and area thorough each geometrical region and feature of the flow path, including tubes/channels, bends, expansions, and contractions.
Equation:
$$\frac{\rho_{exit} V_{exit}^2}{2} = g(H_c + W_c + H_{sh})(\rho_{amb} - \rho_{exit}) - \sum_l \rho_l \frac{V_l^2}{2}\left(\frac{f_l x_l}{D_{h,l}} + K_l\right)$$</td></tr>
</table>

Table (3.2 contd.). Component models with their inputs and outputs.

$\boldsymbol{T}, \boldsymbol{A}_{flow}, \boldsymbol{D}_h, \dot{m}_{gas}, H_c, H_p$ $\boldsymbol{A}_{fuel-bed}, \emptyset_{flame}, \emptyset_{gas}, \boldsymbol{T}_{pot}, \boldsymbol{T}_{firebed-pot}, \boldsymbol{\varepsilon}_{char},$ $\boldsymbol{T}_{int}, \boldsymbol{T}_{ext} \boldsymbol{T}_{wall}$	$\dot{m}_{gas-new}$
MODEL 6 – l_2 norm between mass flow rates. **Equation:** $\varepsilon = \left\| m_{gas}^{p} - m_{gas}^{p-1} \right\|$	
$\dot{m}_{gas}$, $\dot{m}_{gas-new}$	l_2 norm value
MODEL 7 – Check Determines if the iterations need to proceed. If not, sets a flag in the message that is read by the FMS which stop scheduling this system of models any further.	
l_2 norm value, ε	$true \,/\, false$

Each component model was re-factored with the Model SDK and assigned a unique channel ID as explained in section 3. Once the model services are started, they are considered to be deployed and ready to process messages routed to them by the FMS. Figure 3.9 shows the integrated system using the component models. It can be seen that the time loop encompasses most of the models except for the meshing model. Each of the models are invoked in the order as shown and state passed via the message contract from one model to the next. For every iteration, the l_2 norm is computed and the norm is passed to the "Check" model, which checks the norm with the user specified convergence criterion. If this criterion is satisfied, the "Check" model updates a flag in the message. When this message is received by the FMS, it checks this flag to determine if further iterations are needed. If so, they next step in the iteration of the models is invoked. If not, the FMS does not schedule any further models and the final message which contains the stove efficiency is returned to the caller for proving the final answer to the user.

The user inputs are the parameters that allows a user to evaluate the efficiency of a cookstove design and are discussed in detail in MacCarty and Bryden, 2015. In this work the user inputs are provided to the federated system via a web API end-point. This can be enhanced to provide a web-browser based GUI, using which the cookstove design parameters can be input to the federated system model by the user. The output efficiency from the distributed system of models was compared with the monolithic code for the same given inputs. These results are shown in Table 3.4.

In each of the three cases, the resulting efficiency value was identical using the monolithic model and the system of component models. This demonstrates that the decomposition of models did not affect the outcome of the computations.

In Table 3.4 it can also be observed that the computational time is higher in the cases where the federated models are used. This is expected in due to network latency, in communicating between the FMS and the component model services. Additionally, as noted in MacCarty, 2015, the models utilized in this work have been built for preliminary design, hence the computational time is low compared to models built for detailed design, e.g. CFD models. Additionally, since the results of the federated system of models is accessed through a web-browser, it added on average about 100 milliseconds to the response time. The monolithic code on the other hand, reports the final efficiency as console output.

However, it is important to note that the monolithic code does not support easy evaluations, especially in the case where multiple design parameters need to be evaluated. Input parameters would have to be input from either a file or using a script based method.

Table 3.3. Design variables for the cases.

Case	Dc	Hc	Gc	Hp	Ds	Ks	Hsh	Tsh	Ksh	gp	gs	Dp
1	0.1	0.2326	0.025	0.11052753	0.101	1.0	0.08	0.0005	35.0	0.01	0.008	0.24
2	0.4	0.2326	0.25	0.11052753	0.301	15.0	0.08	0.005	3.0	0.01	0.008	0.34
3	0.2	0.2726	0.25	0.110	0.301	2.0	0.08	0.005	30.0	0.01	0.008	0.24

Table 3.4. Efficiency and time comparison of monolithic model with the system of models.

Case	Efficiency (%)		Time (seconds)	
	Monolithic model	**System of models**	**Monolithic model**	**System of models**
1	34.7	34.7	2.2	3.06
2	13.69	13.69	1.06	2.75
3	28.4	28.4	0.72	1.33

Furthermore, once a system model has been established it does not need to be disassembled. As the design moves from the preliminary phase to the detailed-design phase, simple models can be replaced by detailed models. Similarly, during the design optimization

phase, the detailed models can be replaced by ROMs to support quick function evaluations of the optimization algorithm objective function.

3.8. Conclusions and Future Work

Lloyd et al. (2011) classified environmental modeling frameworks as "traditional vs. lightweight" and presented a methodology for measuring framework "invasiveness," defined as the "degree to which model code is coupled to the underlying framework." In this article the novel concept of a federation system of models was proposed and implemented. A federated system of models were developed from a monolithic model code, for the preliminary design of cookstoves.

Another interesting extension of this work would be to use machine learning techniques to automatically learn the characteristics of individually engineered systems (topological, operational, response characteristics, etc.) and intelligently suggest to the FMS the extents of composability. Fusion of computational information with experimental or on-field observations is yet another open question in the field of model integration. Stringent access controls pertaining to user specific security protocols will have to be adapted in order to make this technology useful in an enterprise.

References

Laniak G. F., Olchin G., et al., 2013. Integrated environmental modeling: A vision and roadmap for the future, Environmental Modeling and Software, vol. 39, pp 3-23.

Muth Jr., D. J., Bryden K. M., 2012. An integrated model for assessment of sustainable agricultural residue removal limits for bioenergy systems, Environmental Modeling and Software, vol. 39, pp 50-69.

Allan B., Armstrong R., Lefantzi S., Ray J., Walsh E., Wolfe P., 2005. Ccaffeine: a CCA component framework for parallel computing. Common Component Architecture Forum. http://www.cca-forum.org/ccafe (Retrieved 15 Feb 2016).

Peckham, Scott D., Eric, W. H. Hutton, and Norris, Boyana, 2013. "A component-based approach to integrated modeling in the geosciences: The design of CSDMS." Computers & Geosciences 53: 3-12.

Arnold, T. R., 2013. Procedural knowledge for integrated modelling: towards the modelling playground. Environ. Modell. Softw. 39, 135–48.

Rizzoli, A. E., Leavesley G., Ascough II J. C., Argent R. M., Athanasiadis I. N., Brilhante V., et al., 2008. Integrated Modelling Frameworks for Environmental Assessment and Decision Support. In: Jakeman, A. J., Voinov, A. A., Rizzoli, A. E., Chen, S. H. (Eds.). Environmental Modelling, Software, and Decision Support: State of the Art and New Perspectives, Vol. 3, Elsevier, Amsterdam.

Scientific Computing and Imaging Institute (SCI Institute), 2016. SCIRun. http://www.sci.utah.edu/cibc-software/scirun.html (last accessed 15 Feb 2016).

Vöckler J., Juve G., Deelman E., Rynge M., and Berriman G. B., 2011. Experiences Using Cloud Computing for a Scientific Workflow Application, Proceedings of the 2nd international conference on Scientific Cloud Computing, pp 15-24.

Bernholdt D.E., Allan B.A., Armstrong R., Bertrand F., Chiu K., Dahlgren T.L., et al., 2006. A component architecture for high-performance scientific computing. Int. J. High Perform. Comput. Appl. 20, 162–202.

Jorissen K., Villa F.D. and Rehr. J. J., 2012. A high performance scientific cloud computing environment for materials simulations, Computer Physics Communications, vol 183, issue 9, pp 1911-1919.

Zhao. Y., Li Y., Lu S., Raicu I., et al., 2014. Devising a Cloud Scientific Workflow Platform for Big Data, 10th Word Congress on Services, pp 393-401.

Iosup A., Ostermann S., Yigitbasi M. N., et al., 2011. Performance Analysis of cloud computing services for many-tasks scientific computing, IEEE Transactions on Parallel and Distributed Systems, vol. 22, pp 931-945.

Fox A, 2011. Cloud computing – What's in It for Me as a Scientist, vol 331, Science, pp 406-407.

Belgacem M. B, Chopard B, 2015. A hybrid HPC/cloud distributed infrastructure: Coupling EC2 cloud resources with HPC clusters to run large tightly coupled multiscale applications, Future Generation Computer Systems, vol. 42, pp 11-21.

Thönes J., Microservices, 2015. IEEE Software, vol. 32, no. 1, pp. 116-116.

Michel J. P., Web service APIs and Libraries, American Library Association, 2013.

Villamizar M., et al., 2015. Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud, Computing Colombian Conference (10CCC), 10th, Bogota, 2015, pp. 583-590.

Pahl C. and Lee B., 2015. Containers and Clusters for Edge Cloud Architectures -- A Technology Review, Future Internet of Things and Cloud (FiCloud), 3rd International Conference on, Rome, 2015, pp. 379-386.

Armbrust Michael, et al., 2010. "A view of cloud computing." Communications of the ACM 53.4: 50-58.

Suram S. and Bryden K. M., 2015. Integrating a Reduced-Order Model Server into the Engineering Design Process, Advances in Engineering Software, 90:169–182.

MacCarty N. A. and Bryden K. M., 2015, Modeling of Household Biomass Cookstoves: A Review, Energy for Sustainable Development, 26:1–13.

Johnson+ P. E., Ashlock D. A., and Bryden K. M., 2014. A Novel Engineering Tool for Creative Design of Fluid Systems, Engineering with Computers, 30(1):15–29.

MacCarty N. A. and Bryden K. M., in review submitted December 2015, A Generalized Heat-Transfer Model for Shielded-Fire Household Cookstoves, Energy for Sustainable Development, in review.

Bryden K. M., 2014. A Proposed Approach to the Development of Federated Model Sets, Proceedings of the 7th International Congress on Environmental Modelling and Software.

International Energy Agency (IEA), 2010. Energy Poverty: How to make modern energy access universal?, World Energy Outlook, Paris.

Lloyd W., David O., Ascough II J.C., Rojas K.W., Carlson J.R., Leavesley G.H., et al., 2011. Environmental modeling framework invasiveness: Analysis and implications. Environ. Modell. Softw. 26(10), 1240–1250.

Ascough II J.C., Flanagan D.C., David O., 2005. Assessing the potential of the object modeling system (oms) for erosion prediction modeling. In: Proceedings 05-211. 2005 ASAE Annual International Meeting, Tampa, FL.

David O., Markstrom S. L., Rojas K. W., Ahuja L. R., Schneider I. W., 2002. The object modeling system. In: Ahuja, L., Ma., L., Howell, T.A. (Eds.), Agricultural System Models in Field Research and Technology Transfer. Lewis Publishers, Boca Raton, FL, pp. 317–331.

Rahman J. M., Seaton S.P., Perraud J. M., Hotham H., Verrelli D. I., Coleman J. R., 2003. It's TIME for a New Environmental Modelling Framework. In: Post, D.A. (Ed.). MODSIM 2003 International Congress on Modelling and Simulation, 4, 1727–1732.

Gregersen J. B., Gijsbers P. J. A., Westen S. J. P, 2007. OpenMI: Open modelling interface. Journal of Hydroinformatics, 9(3): 175–191.

Blind M. and Gregersen J. B., 2005. Towards an Open Modeling Interface (OpenMI) and the HarmonIT project. Advances in Geosciences, 4: 69-74.

MathWorks. 2016. Simulink – simulation and Model-Based Design. 2011. The MathWorks, Inc. http://www.mathworks.com/products/simulink/ (last accessed 15 Feb 2016)

Simulia™, 2016. Execution Engine (formerly Fiper). http://www.simulia.com/products/see.html (last accessed 15 March 2016).

Phoenix Integration, 2008. PHX ModelCenter® 10.0. http://www.phoenix-int.com/ (Retrieved 15 March 2016).

Pro-Trax Suite Simulation Software, 2016. http://www.traxintl.com/ (last accessed 10 March 2015).

MacCarty N. A., Bryden K. M., 2015. Modeling of household biomass cookstoves: A review. Energy for Sustainable Development 26:1-13.

Armstrong C. W., Ford R. W., Riley G. D., 2009. Concurrency and Computation- Practice & Experience, 21: 767-791.

Balaji V., 2002. The FMS Manual: A developer's guide to the GFDL Flexible Modeling System. URL: http://www.gfdl.noaa.gov/fms (Retrieved 18 January 2015).

Redler R., Valcke S., Ritzdorf H., 2010. Geoscientific Model Development 3: 87- 104.

Bryden K. M., and McCorkle D. S., 2004. "VE-Suite: a foundation for building virtual engineering models of high performance, low emission power plants." 29th International Technical Conference on Coal Utilization & Fuel Systems, Clearwater, Florida. R. Perrey and M. Lycett, "Service-oriented architecture," Applications and the Internet Workshops, 2003. Proceedings. 2003 Symposium on, 2003, pp. 116-119.

Amazon, Overview of Amazon Web Services, https://d0.awsstatic.com/whitepapers/aws-overview.pdf, Retrieved Feb., 2016.

Fielding Roy T., and Richard N. Taylor, 2002. Principled design of the modern Web architecture, ACM Transactions on Internet Technology, 2.2: 115-150.

Erl T., 2005. Service-Oriented Architecture (SOA): Concepts, Technology, and Design, Prentice Hall.

Giessmann, Andrea, Stanoevska-Slabeva, Katarina, and Visser, Bastiaan De, 2012. Mobile Enterprise Applications--Current State and Future Directions, System Science (HICSS), 2012 45th Hawaii International Conference on. IEEE.

Nolan D., and Duncan T. L., 2014 Javascript object notation, XML and Web Technologies for Data Sciences with R. Springer New York. 227-253.

McNunn G. S., and Bryden K. M., 2013. "A Proposed Implementation of Tarjan's Algorithm for Scheduling the Solution Sequence of Systems of Federated Models." Procedia Computer Science 20: 223-228.

Babel, 2016, http://csdms.colorado.edu/wiki/Help:Tools_CSDMS_Handbook (Retrieved 25 March 2015).

Microsoft, 2016, https://azure.microsoft.com/en-us/solutions/big-compute/ (Retrieved 25 March 2016).

CHAPTER 4. A NOVEL APPROACH TO INTEGRATE A COMPONENT ROM INTO A DISTRIBUTED ENGINEERING SYSTEM MODEL

Article to be submitted to *Advances in Engineering Software*

Sunil Suram and Kenneth M. Bryden*

Abstract

As computational models and simulations are getting easier to run with the advent of cloud computing, the management of the associated data and models is getting more difficult. For producers of these high-fidelity models getting access to high-end hardware has become easier. However, for consumers of the models, key stakeholders in the design process, and designers, the process of utilizing these models in decision-making tasks the complexity involved has not decreased by increases in computational power. A novel engineering workflow based approach is proposed in this article to bridge this gap using information artefacts. In the proposed approach, data from high-fidelity computational models are utilized to construct ROMs seamlessly without user intervention. Utilizing the ROMs and the computational models concurrently, a higher level of abstraction to these models is created as an information artefact. The consumers of this information can query the information artefact for information based in the design parameters. The information artefacts are web-enabled and communicate with a federation management system. This approach is demonstrated using a heat-exchanger fin shape design example and comparisons are drawn between the resulting engineering workflow and the workflow proposed in Suram and Bryden, 2015. It is found that this proposed approach has the

potential to make the consumption of engineering modeling information easier by removing the tight coupling between the producers and consumers of these models.

4.1. Introduction

In a traditional modeling approach, the individual component models are linked with each other using software/code. Each of the models is usually a software library that exposes application programming interfaces (APIs). All the linking occurs in a single software program that brings together the individual models, including problem specific entities like initial conditions, boundary conditions, geometry and information/data transfer between models. More often than not, the monolithic software program also incorporates elements of the hardware that it is going to run on. For example, it can make assumptions about the availability of certain compute clusters, parallel programming models etc. Although this approach can be effective in solving the problem at hand, it makes it difficult for another engineer to reuse this work without undertaking the effort of refactoring the code to suit the new problem to be solved.

One way of overcoming this problem of building and modeling large-scale models is by taking an integrated framework based approach, where each individual model is part of a larger framework of models (Peckham, Hutton and Norris, 2013). The overall framework establishes bounds on model developers wherein each model has to satisfy a set of criteria to fit into the integrated modeling framework (Peckham, Hutton and Norris, 2013). An advantage of such an integrated framework is that all the individual models are consistent. Once a model developer understands the framework, developing newer models that are consistent with the established protocols of the framework becomes easier. On the other hand, imposing functional model development constraints on developers at a global level is a difficult task. Several groups of developers may consider this too restrictive and suggest changes to the integrated framework. As

a result, the integrated modeling framework changes with time and can cause version compatibility issues. Thus such a centralized approach can be useful for smaller groups of engineers, but quickly becomes intractable for universal adoption.

4.1.1. Reduced Order Models

Engineering design is an iterative decision making process in which collaborative groups of designers/engineers work together from conceptual design to a final product. Many engineering design workflows have been proposed but most of these are similar to Figure 4.1 (Pahl et al., 2007; Ertas and Jones, 1996). As shown in Figure 4.1, the design process is composed of three main stages, (1) problem definition, (2) engineering design, and (3) design validation and verification. (Pahl et al., 2007; Ertas and Jones, 1996).

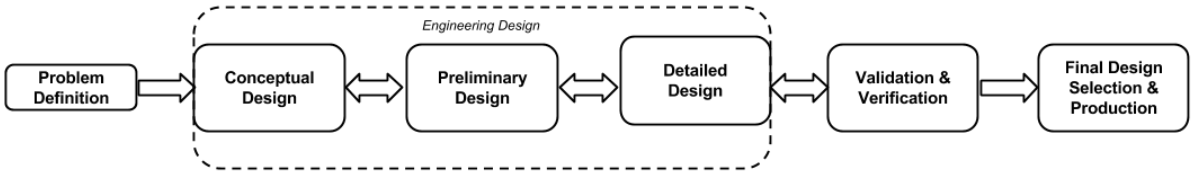

Figure 4.1. Workflow in engineering design.

During the detailed design phase the chosen design is optimized and finalized. The engineering design stage involves an iterative process in which the designers typically move back-and-forth between conceptual, preliminary, and detailed design with relatively quick consideration and analysis of multiple designs, repeated iterations and expansions of proposed solutions, revisiting assumptions and decisions, and a series of design changes. Once completed a reduced set of designs are chosen for further verification and validation using additional analysis and field data. This can lead to changes to the initial design, thus requiring a repeat of the design stage. The exploratory and iterative nature of engineering design makes the process of communicating engineering information and analysis during the design stage, between various engineering teams challenging. Suram and Bryden, 2015, introduced a ROM server that stores all the data from computational models and creates reduced order models from the data. The ROM server enhanced communication between analysts and designers compared to the traditional model of running ROMs. However, in cases where the ROM server does not have sufficient data, the communication between the analysts and designers has to be explicit, in order to update the data. To further streamline this process, in this research the concept of information artefact is introduced. Using information artefacts the boundary between a detailed model and a ROM is obfuscated helping make the engineering decision making process.

4.1.1.1. Proper Orthogonal Decomposition

In this work the proper orthogonal decomposition (POD) technique is used to construct and evaluate ROMs. The POD technique has been used extensively in prior research and details about can be found in (Suram and Bryden, 2015). POD has been incorporated into the engineering workflow as described in (Suram and Bryden, 2015). The POD technique is useful because it captures all the required information about the phase space of a given physical

problem. When using it to solve an engineering design problem, this information can then be used in conjunction with the coefficient interpolation technique to explore the design space in a computationally efficient manner. This process can be summarized as follows:

- Identify the design parameters and design space of interest
- Create the computational data needed for the snapshot dataset that spans the design space of interest
- Create the POD coefficient and basis functions
- Make the POD ROM available for use
- Use the POD ROM to compute new solutions as needed to support the engineering design process

If the design space to be explored needs to be expanded or new aspects of the problem need to be explored, the snapshot dataset will need to be expanded and a new POD ROM will need to be developed. Additionally, the accuracy of the POD ROM increases as the number of snapshot solutions increases. Thus as the design process evolves and more accurate solutions are needed, the POD ROM will likely need to be updated in regions of the design space of particular interest.

The iterative nature of the design process and the continuing update of the ROM creates a communication challenge within the design and analysis team. To evaluate a ROM, the most recent set of coefficients and the basis functions need to be known by the user. If a user is geographically in a different region or a part of a different engineering team interested in evaluating the ROMs or analyzing the results, this information has to be made available to them. Updating the ROM manually via email or download for local compute makes it challenging to ensure that the most recent ROM is used and that disparate members of the design group are

using the same ROM. Furthermore, providing local access to this data for multiple users may not be possible. It is also likely that multiple POD ROM models would be used to address a large-scale complicated design problem, and a process is needed to coordinate the development and use of these multiple POD ROM models. This creates problems with management of data and version control. The ROM may remain on a single computational machine or may be exported to remote machines for simultaneous use. If it is kept on a single machine, access is limited because only one ROM computation can be performed at a time. If it is exported, maintaining version control of the ROM becomes difficult and different groups having conflicting or out-of-date information can slow the design process. In the next few sections we propose an engineering workflow to overcome these challenges and enable the seamless utilization of ROMs within the engineering design process.

4.1.2. Information Artefacts

In this article the concept of information artefacts is introduced. For an engineering system, an information artefact is a provider of information to a system model i.e., they can be computational models, closed-form solutions, data, design parameters, optimization algorithms, etc. In this context, all these pieces of information are recognized as information artefacts (IAs) where they can be queries with an input and a response of returned from them.

Information artefacts are a higher-level of abstraction from detailed models and ROMs i.e. they encompass the information within each of the models. Thus when an information artefact is invoked by the FMS, either the detailed model or the ROM can be invoked in-turn, based on how much information each of these models contains. If the ROM has insufficient information, the detailed model is invoked. The advantage of this abstraction is that for tasks that

do not require detailed models, a user cannot inadvertently invoke a model of higher computational cost.

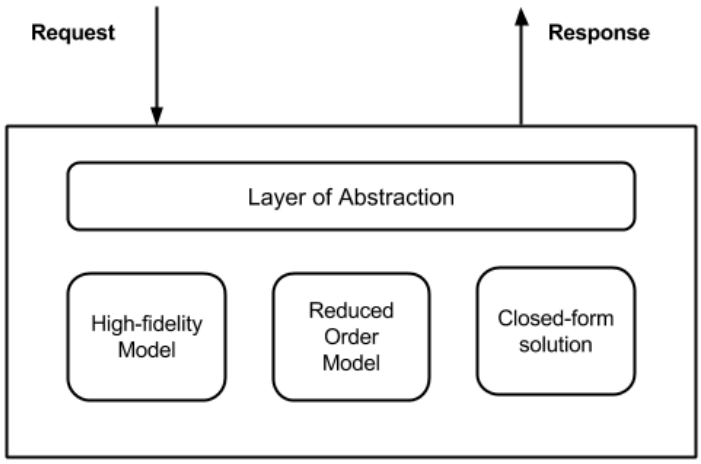

Figure 4.2. Information artefact.

As shown in Figure 4.2, when a user requests a computation from the information artefact there is no distinction made between the high-fidelity model and the ROM. The model that is capable of returning the information fastest is given priority. If a detailed model solution to the request exists in the cache, it is returned immediately without performing a ROM computation. On the other hand, if a detailed model solution does not exist, a ROM computation is performed. If both these criteria are not satisfied, the information artefact proceeds to perform a detailed model computation. Similarly, it is known that closed-form solutions to complex problems are not always possible, especially for complex boundary conditions. However, can it be used within an information artefact during preliminary design to understand the response of an engineering system model in order to inform some engineering decisions. In this article, the terms

information artefact and hybrid model are used interchangeably, since the information artefacts used in this work are limited to computational models and ROMs.

4.1.2.1. Model Substitution

In this work, the framework developed in (Suram and Bryden, 2016) is extended to include the concept of engineering model substitution. There are several scenarios during the engineering design process where complex models have to be substituted by simpler models. This is typically needed when "what-if" analysis needs to be performed to study and analyze the impact of certain design decisions. In such situations, running high-fidelity models is an option but involves a high turnaround time for completion. When multiple design iterations are involved, high-fidelity models increase the time to make the engineering decision.

In this article, a framework is outlined for these scenarios where a detailed high-fidelity model can be substituted by a simpler model that has a faster computational time, but has the ability to provide the same output for the engineering system model. The framework is capable of storing high-fidelity computational data and automatically creating reduced order models utilizing the proper-orthogonal decomposition technique.

4.1.3. Federated Models

In the framework developed in (Suram and Bryden, 2016), a collection of individual models constitutes a federated set of models, where each of the models has a set of inputs and outputs. It was demonstrated that a complex system model can be built by composing together multiple individual models and can be deployed as a system model that can be reused and exposed as an Internet application (Suram and Bryden, 2016). Furthermore, any of the developed models in the federation set do not have any schema imposed on the structure of inputs and outputs. The model developer decides the structure of inputs and outputs that the model accepts

and emits, the only requirement being that this information be broadcast to model developers via API calls. The communication between constituent models is orchestrated by a federation management system (FMS). The FMS is aware of all the models that are registered with it and has the ability to trigger the execution of any of the registered models as needed. A user communicates with the FMS by initializing the desired model, describing a list of individual models that constitute it and supplying the boundary conditions. The FMS is responsible for the communication between each constituent model. The FMS is also responsible for the availability of computational resources to execute any registered model by starting up new instances of a model based on usage.

For such a decentralized system of models to be functional and scalable a cloud-based architecture is the most practical solution. The primary reasons for this are:

- Universal availability: Cloud platforms can be accessed by anyone with a web-browser and an Internet connection. This opens up the possibility for model developers to build and publish their models either as part of a closed group or globally regardless of their geographic location. Model developers should be able to publish their models by registering them with the FMS.
- Scalable platforms: As more models get added computational resources need to be increased automatically without human intervention. The deployed models need to be readily available every time the FMS sends a request for computation. Cloud platforms are inherently scalable in terms of hardware and the process of scaling can be made intelligent and automated by utilizing platform level APIs from the providers. Thus, highly scalable and fault-tolerant software systems can be developed.

- Cost Effective: Cloud platforms work on a cost per usage model and are hence cost effective as the user only pays for the compute time and not for procuring, provisioning and maintaining compute, storage and networking resources.

4.1.4. Loosely Coupled Engineering Models

In traditional approaches to linking engineering models (Babel, 2016; Peckham et al., 2013) the "linkage" occurs in code i.e. information transfer between models has to be incorporated in code. Thus the model-associated state information is tightly coupled to the models being used and to the execution context of the engineering problem. This makes the coupled code difficult to reuse without modifications and refactoring because of the state information is embedded into this coupled model. A primary reason for this is the availability of models as software libraries as opposed to individual atomic compute "engines" which act on request to a specified set of boundary or initial conditions.

Such an implementation would require a loosely coupled distributed system based approach to building each model, scheduling their execution and orchestrating information transfer. An important requirement is that state information be moved away from individual models as they run. This enables any model of the same type can carry out subsequent runs without keeping track of state information. This is an important requirement when linking models together, because information transfer from one model to another can occur by models loading the updated state at run-time. This is an important distinction that the authors would like to note, from a "library approach" where this coupling occurs primarily at compile time of the code. This also plays a key role towards enabling a fault-tolerant and scalable distributed system to be constructed of collections of engineering models. Moving the contextual state information away from a model helps make it a foundational construct within a larger collection, which can

be called to perform a task. Existing cloud computing platforms can be used to "spin-up" new model instances with increases and usage and demand requirements. When engineering models are coupled using code, they become tightly-coupled and become difficult and unwieldy to take apart for reuse. For this reason, if models are built as individual functional units that perform a particular computational task.

4.1.5. Stateless Models

The concept of statelessness is critical to the implementation of federated model sets described here. State refers to the entirety of information that defines a model while executing a computational task. For example, in an RK4 model, the initial time-step, initial conditions, constants in the evolution equation and the current time-step define the state of the RK4 model. If state within a model is continued to be maintained beyond the execution time-frame, state is said to persist. This is typically the case in monolithic computer codes (system models) where state in one model is continued to be maintained while a different model is executing based on the state from the first model. However, if the model does not retain any state between invocations it is said to be stateless (Thönes, 2015). If it is possible for the model to pass-on its state information to the next model without persistence, the individual models themselves become reusable computational entities across systems and multiple system models. Any system model can use an individual model as needed. As noted above, this work requires that each individual model is stateless i.e. the model does not persist state information beyond the executing time-frame for the current task. This is an important consideration because it enables each model to act as a "functional unit" that can be reused and combined easily with other models. It must be noted that stateless models can be created either as a single solver, say, an RK4 solver or it could be a combination of multiple solvers. Participation within a federation of

independent models requires that each model be stateless and that each model identify its inputs and outputs. An important aspect of being stateless is that the message and hence the code can be executed on any node in the federation, which is an important aspect of scalability.

4.2. Workflow

4.2.1. Engineering Workflow Using the ROM Server

Figure 4.3 shows the engineering workflow using the ROM server introduced in previous work (Suram and Bryden, 2015). Engineering design is a complex and iterative process that involves multiple engineering teams sharing and communicating information during the design process. Computational modeling and the development of high-fidelity models play a significant role in the design process. High-fidelity models are accurate but on the other hand they are time-consuming and can slow down the design process. To address this, a framework was developed in (Suram and Bryden, 2015) to integrate ROMs into the engineering design workflow. As was shown in Suram and Bryden, 2015, the ROM server is capable of providing a single consistent view into the computational models within an organization. However, for the ROM server to function consistently, the producers and consumers of models must communicate outside the context of this information space. For example, adding, updating or deleting data from the ROM database requires that the producers and consumers of the model communicate with one another. Thus, the ROM server implemented in Suram and Bryden, 2015, requires an implicit coupling between the producers and consumers of models and does not enable seamless interactions between models, data and the associated analysts and designers.

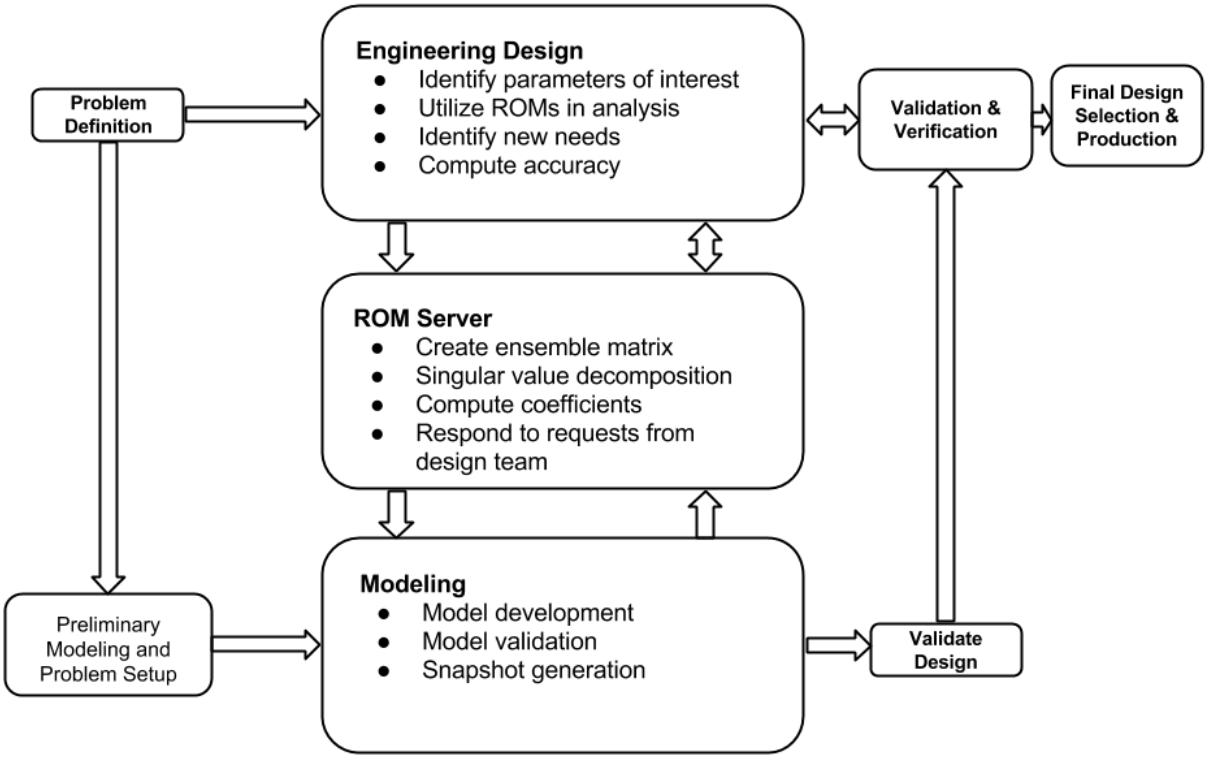

Figure 4.3. Workflow developed utilizing the ROM server.

To enable such seamless interactions between all the associated stakeholders of an engineering models, within an organization, requires a framework that makes use of the concept of information artefacts. It should allow producers (analysts) to produce high-fidelity models and the consumers (designers) to interact with the information artefact, as opposed to the ROM server. It must thus enable dynamic and seamless substitution of high-fidelity models with reduced order models. As a part of this, the goal is to build the software infrastructure to enable such model substitution and reuse. These models can include engineering models like finite element analysis and computational fluid dynamics as well as non-engineering models including cost models and product diffusion models. In this work we build on the approaches developed in Suram and Bryden, 2015 and 2016, to include reduced order models that can substitute high-fidelity models to form a linked system of models that solves a larger problem. This approach requires

1. A set of models that solve specific problems. These models can be self-contained individual solvers that implement a specific algorithm or can be composed from other existing models. Users must have the ability to add more models to the federation as they see fit.
2. A management system that can accept new models and broadcast the details about existing models in the federation set. This federation management system (FMS) needs to be able to accept requests from users and orchestrate, coordinate and execute models in the prescribed order necessary to solve the problem.
3. A framework of communication between models and the FMS where each model can receive instructions to execute and notify the FMS of the completion of a computation or of errors.

In addition, the framework also needs the following services to perform computations and store the generated data for reuse.

1. A cloud-based data storage layer that can store high-fidelity models efficiently in a scalable manner. The results from the high-fidelity models can be cached for reuse with the same input design parameters.
2. A generic model that is capable of generating ROMs using the data generated from the high-fidelity computational models and input design parameters.
3. A generic model that uses the above generated ROM and evaluates user requested input design parameters and returns the output to the user. This is useful in order to reduce computational time for a set of design parameters that lie within the phase-space of the problem being considered. The user must be allowed to assign a higher preference to either the accuracy of the computation or the time to solution. In the former case, the high-fidelity computational model is invoked by the FMS and in the latter the ROM evaluation is invoked.

4.3. Improved Workflow

In this article, an improved engineering workflow is proposed utilizing information artefacts. The previous section summarized the engineering workflow from Suram and Bryden, 2015, based on the ROM server. IAs allow the seamless integration of ROMs and high-fidelity models. Thus, IAs can be a starting point for the integration of models as well as the interactions between analysts, designers, and engineering decision-makers. Figure 4.4 shows the proposed, improved workflow where the IA is central to all the interactions between the producers and consumers. Additionally, the IA is also central to the process of collecting computational data and computing ROMs from the data.

The producers, during the design process formulate the problem, identify design parameters of interest and run high-fidelity models. The data from these models is stored in the background without any user intervention or explicit instructions. As the data gets collected, the ROM construction service periodically constructs ROMs from this data and updates the ROM parameters. The designers can simply query the IA for information about the design, which responds with the solution based on either the ROM or the high-fidelity model. The IA uses the "energy" of the ROM to determine if sufficient data was used in its construction.

It can thus be seen that this proposed workflow improves the engineering workflow (Suram and Bryden, 2015) by eliminating the need for interactions between designers and analysts. For updates to the design during the design process, each analyst and designer can proceed with their own tasks, and the IA will process the requests either using the ROM or the high-fidelity model. The remainder of this section provides details about the methodology and architecture built to develop the information artefact.

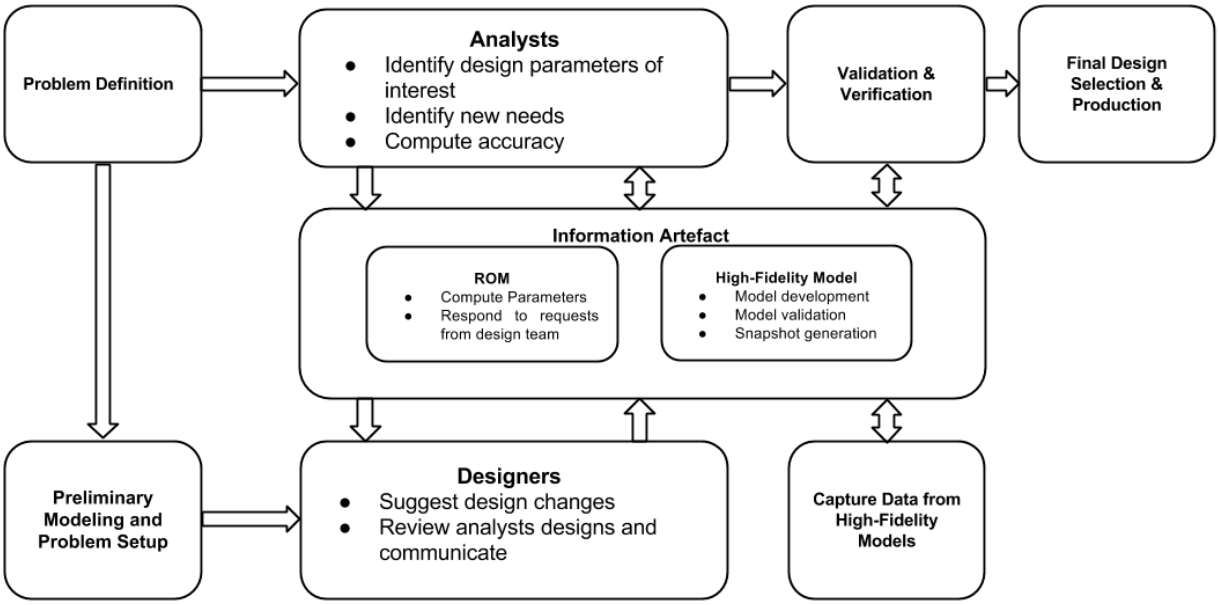

Figure 4.4. Improved workflow with the information artefact.

4.3.1. Methodology

To address the problems described in the previous section the framework developed by Suram and Bryden, 2015, is extended. The key additions to the framework are the

a. ROM construction model

b. ROM evaluation model

These models provide the system builder the option of generating ROMs for computationally expensive models and utilizing them during the design process when quick iterations on the design are needed.

In Figure 4.5, the overall architecture of the distributed system for model substation is shown. There is an association between Model 1 and its corresponding ROM and similarly between Model 2 and its corresponding ROM. In both cases, the model and the ROM accept the same set of inputs and their corresponding outputs are the same and hence they can be substituted for one another. For example, the system builder can invoke the ROM for a set of

input design parameters instead of running the high-fidelity model. However, this substitution cannot occur in all situations. For example, the user might be requesting an evaluation of the ROM, say, for Model 2 with certain input parameters that were not within the initial design parameter space used to construct the ROM. In such situations, the FMS must make a decision about the high-fidelity model that needs to be invoked for this set of input parameters. For a decision to be made there must be established rules that help the FMS make an appropriate decision wherein the high-fidelity model is invoked.

4.3.2. Model Substitution and Evaluation Rules

A key aspect of this work is the ability of the FMS to seamlessly substitute a high-fidelity computational model with a ROM. To enable this substitution, the FMS needs to have access to key pieces of information concerning the high-fidelity model. These are:

1. The design parameters
2. Access to computed data from these models
3. A set of rules that the FMS can refer to when making the decision of switching between the high-fidelity and reduced order models. This requires association rules that establish substitutability of a high-fidelity model with a ROM.

To further explain association rules for substitutable models, a simple example is provided. Consider the computational problem of solving the Poisson equation in a square domain as shown in Figure 4.5 along with Dirichlet boundary conditions on all four boundaries. Only the top boundary has a variable parameter boundary condition while the other three sides do not vary. For this simple problem, the top boundary temperature is the input design parameter and the designer's goal is to compute the temperature field in the domain for any acceptable design parameter value.

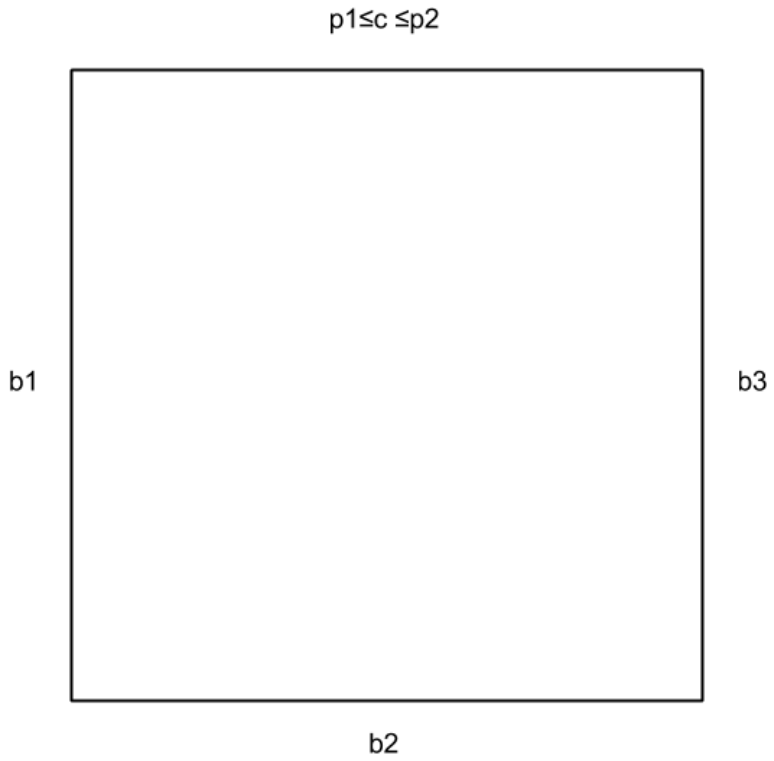

Figure 4.5. Poisson equation on a square domain with boundary conditions.

A computational model that is based on solving the equation () numerically is considered, in this example, as the high-fidelity model. Using data from this model, a ROM is constructed that takes the same input parameter. It must be noted that the ROM evaluations are valid only in the parameter range shown in the Figure 4.5. For any values of the input parameter *p*, that are outside the range *[p1, p2]* the computational model will have to be invoked to solve the problem. Based on these criteria, either the numerical model or the ROM can be evaluated for an input design parameter. Thus, the federation management system needs to be able to use association rules set by the system builder and invoke the appropriate model based on the user preferences of accuracy or speed.

4.3.3. Rules

The system builder creates the association rules that define substitutability of the models. To ensure that incompatible models cannot be associated with one another, the inputs and outputs to and from the models are examined for compatibility. This process can be either programmatic or manual. Once model compatibility is ensured, this information is submitted to the FMS which stores and retrieves it as needed, at run-time. Also, the substitution rules can be added by the system builder dynamically without restarting either the FMS or any of the associated model services. These rules are communicated to the FMS through messages via its input channel (Suram and Bryden, 2016). Table 4.1 shows an example of the constituents of such a message that sets substitution rules between Models 1 and 2 and their appropriate ROMs.

Table 4.1. Contents of an example message that enables model substitutability.

Key	Value
compatible_list	["model1","rom1"],["model2","rom2"]
data_repo	[dfs://model_data_bucket/model1, s3://model_data_bucket/model2]
meta_data_repo	[<table_name>, meta_repo_m1] [<table_name>, meta_repo_m2]
cache_loc	[<model_id>, dist_cache_IP]

The message also contains design meta-data information, which references the parameter values for the design. This key is used by the FMS to lookup the design parameter ranges when there is a ROM associated with a high-fidelity model. This key, points to the location of design parameters which the FMS reads into its own cache. When it receives a user request, the FMS performs a model substitution action based on the user input. If the user has requested accuracy over speed, the FMS invokes the high-fidelity model.

If the user favors speed over accuracy, the FMS will attempt to invoke the ROM. In order to do this, it must a) check its cache for model associativity and find an associated ROM and b) check if the user requested design parameter values are within range of the ROM. If both these conditions are met, the FMS invokes the appropriate ROM. If no substitutable ROM is found or input parameters are out of the range, the FMS reverts to invoking the high-fidelity model and informs the user of this status. The FMS also marks the message with this status, which is read by the FMS when the high-fidelity model execution is complete. The execution of the high-fidelity model proceeds in an asynchronous manner and the user is informed by the FMS when it completes. While informing the user of the completion status, the FMS also sends a message to the ROM construction service to use the newly added data and update the ROM parameters. When the ROM parameters get updated, the ROM evaluation service uses them for processing new requests. The flowchart in Figure 4.6 shows the process that the FMS takes to substitute compatible models for one another.

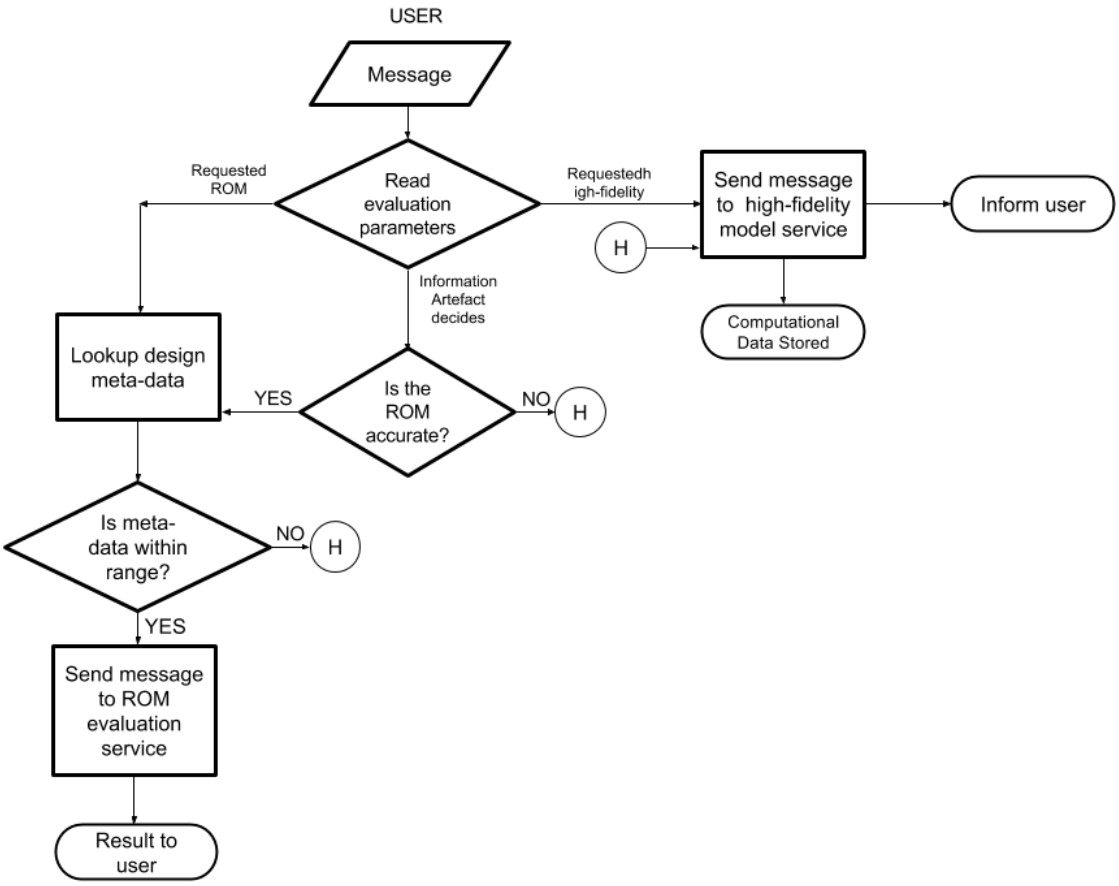

Figure 4.6. Flowchart showing the steps the FMS takes for model substitution.

4.3.4. Data Management

The section outlines the approach taken to manage computational data generated from high-fidelity models in the model federation. Figure 4.7 shows the data flow of computational models and design parameters. The data flow can be divided into 5 distinct steps. The key points to be noted in Figure 4.7 are:

a. As new computational models are created they get added into a repository. Adding data to the repository also triggers the associated model meta-data to be saved in the meta-data repository.

b. This repository is used by the ROM computation service to generate reduced order models.
c. The ROM computation service updates the ROM parameters.
d. The ROM evaluation service utilizes the ROM parameters for fast computations of user inputs.
e. The FMS seamlessly facilitates these interactions and computations without explicit user intervention.

In step 1, data generated by engineering models are added to a model data repository. The model data repository is a cloud based storage platform that stores all the computational data in a fault-tolerant manner. The meta-data associated with this engineering model is stored in the meta-data repository, depicted by step 2. As soon as these steps are complete, the FMS sends a message to the ROM construction service, which uses the data from these two repositories and computes the ROM parameters, as shown in steps 3a and 3b. Since this can be time-consuming, the process is asynchronous and can be scheduled to run on a periodic basis (Suram and Bryden, 2015). The resulting ROM parameters are then stored in a repository for use by the ROM evaluation service as shown in steps 4 and 5. When the FMS can find a compatible ROM, it invokes the ROM evaluation service with inputs of the user requested parameters. The ROM evaluation service uses the ROM parameter repository to compute the output and sends the response back the FMS which responds to the user.

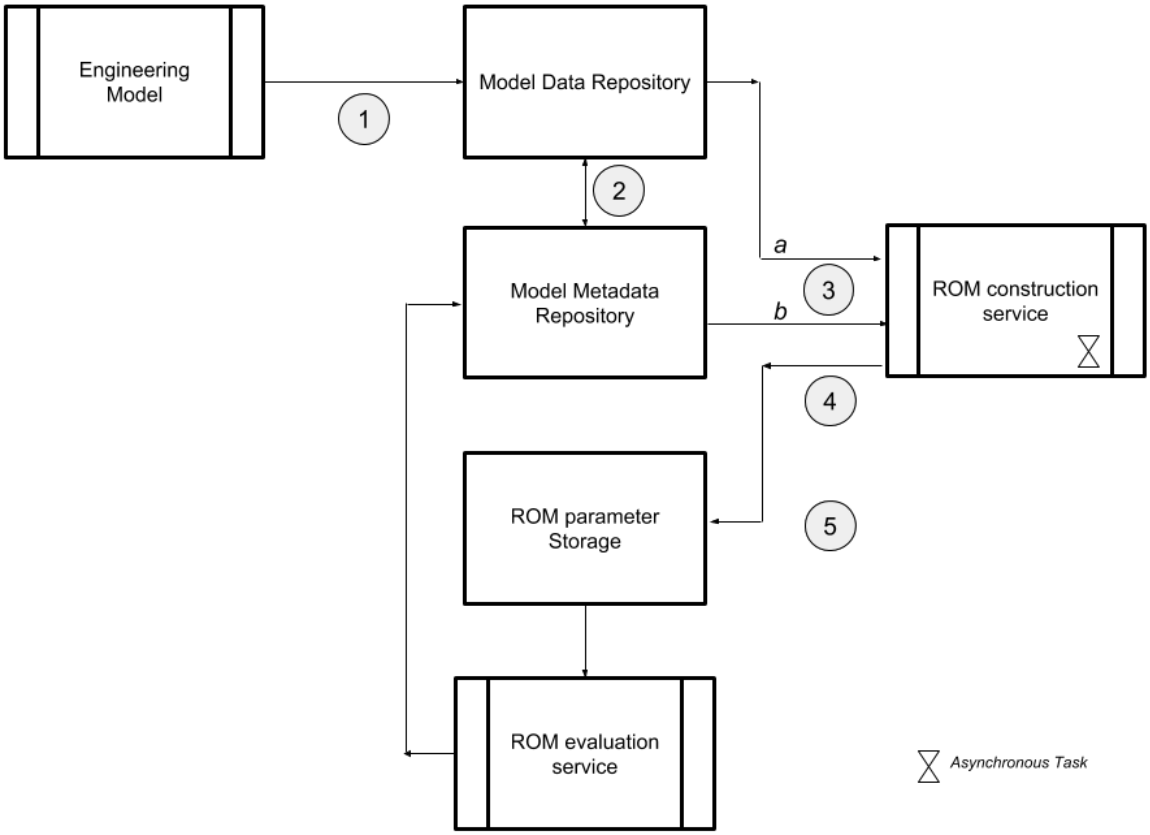

Figure 4.7. Data flow of models and design parameters.

4.3.5. Computations

The computations can be categorized into two types' viz. high-fidelity model and ROM computations. Additionally, the computational time to construct the ROM is more expensive compared to the evaluation of the ROM. The relative computational times of the models considered in this work are shown in Table 4.2. For each category of computation, a different strategy is utilized due to differences in execution times.

Table 4.2. Relative computational time and computation type.

Model Type	Relative Computation Time	Computation Strategy
High-fidelity model	Very high	Asynchronous
ROM construction	High	Asynchronous timed-execution
ROM evaluation	Low	Synchronous

Each of the strategies is explained below in the context of a user request and the model substitution rules given to the FMS.

a. Asynchronous Strategy

Consider a user request for a ROM evaluation that consists of input design parameters that are outside the initial design parameter range of the ROM. In this event, the FMS invokes the high-fidelity model after looking up the substitution rules. Since this occurs without explicit user intervention and the execution time of the high-fidelity model can be very high, the user is immediately notified and the response contains a unique job identifier that can be used to query the FMS of the jobs status. Thus, the execution of the high-fidelity model continues in an asynchronous manner and the user can retrieve the results on its completion.

In the event that the user requested a high-fidelity model, the FMS does not need to lookup substitution rules and proceeds to invoke the high-fidelity model and a job identifier is assigned as described above.

b. Asynchronous Timed-Execution Strategy

This strategy applies to the ROM construction service which uses the results of high-fidelity models to compute ROM parameters. On completion this service updates the

ROM parameters that are used by the ROM evaluation service. Additionally, it also updates the design meta-data that is referenced by the FMS. This computation is dependent on upstream updates to the model database and the downstream ROM evaluation service depends on the results from its successful execution. Also, the ROM construction should not be triggered on every model database update. Considering these restrictions an asynchronous timed-execution strategy is utilized, which triggers the ROM construction service on a periodic basis. Once it begins execution, this service checks for updates to the model database and proceeds to perform a ROM construction only if needed.

c. Synchronous Strategy

Once the ROM parameters are computed, the computation time is small for a user request that is within the parameter range of the ROM. Thus for such a request, the results are returned immediately. Once the FMS looks up the design meta-data, it invokes the ROM evaluation service, waits for the response from the ROM and returns the results to the user. Thus, a single request is sufficient to return the result of a computation to the user.

4.4. Example Application

The shape design of a heat exchanger fin is used in this work as an example problem to demonstrate the capability of the strategy developed in this article to switch between a ROM and high-fidelity model. It should be noted that the problem considered is primarily to demonstrate the value of storing data from high-fidelity models and using the data to a) create ROMs and b) use the ROMs in a manner that is transparent to a user of a system model.

A two-dimensional heat exchanger fin shape design problem has been chosen as the engineering design problem. A brief description of the problem is outlined in this section and further details can be found in Suram et al., 2006. Figure 4.8 shows a set of fins where fluid (water) is pumped through the channel between the curved surfaces of two consecutive fins to remove heat. Four design parameters have been considered:

1. Length of the fin (*a*)
2. Spacing between the fins (*b*)
3. Base thickness (tau)
4. Thickness of the lateral surface of the fin (*y*).

These design parameters are the inputs to the ROM construction service and as well as the high-fidelity model. The outputs from these models are the temperature distribution in the fin and the fluid as well as the velocity distribution in the fluid.

In this example problem, the process of validating the inputs and outputs of the two substitutable models is simple. However, if there are several models the validation process can be programmatic and hence automated. Table 4.3 shows the inputs and outputs from each of the substitutable models. Since the outputs are temperature and velocity fields, for brevity they are depicted symbolically as a numeric array in JSON format. The ROM output contains the energy key-value pair while the output from the high-fidelity model does not. The energy value can be used by the engineer or analyst to determine if the ROM was constructed with sufficient data.

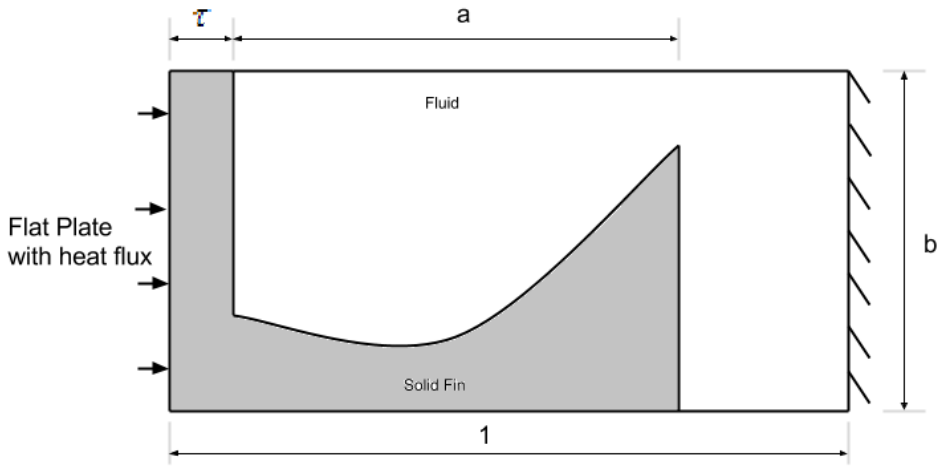

Figure 4.8. A single fin considered in the design problem.

Table 4.3. Inputs and outputs to the substitutable models.

Model	Input	Output

ROM	{
	"a": 1.0,
	"b": 0.224,
	"tau": 0.118,
	"y0": 0.126
	}
		{
		"T": [1,0.99,…],
		"u": [0,0,…],
		"energy": 0.95
		}

High-fidelity	{
	"a": 1.0,
	"b": 0.224,
	"tau": 0.118,
	"y0": 0.126
	}
		{
		"T": [1,0.99,…],
		"u": [0,0,…]
		}

Each of the services are invoked at different points in time and can be categorized into the following timeline:

1. Substitutability rules to FMS
2. Model deployment
3. User requests

4. ROM construction and parameter updates
5. ROM evaluations

These steps are explained in detailed in the following sections.

4.4.1. Substitutability Rules

The first step is to assign substitutability rules to the FMS. In this case, the high-fidelity model and ROM are substitutable and this information is provided to the FMS as shown in Figure 4.2. Once these rules are assigned, the FMS is aware of the compatibility of the models and will attempt to substitute the high-fidelity model with the ROM.

4.4.2. Model Deployment

Once the model developer deploys the model, they are running as background services waiting on the FMS to assign tasks to them, as soon as a user request for evaluation is received. The ROM construction service although deployed, is not triggered to perform any computations since no data has yet been generated by the model services.

4.4.3. User Requests

The data from the first user request for a model bootstraps the high-fidelity model database. Every user request that cannot be evaluated using the ROM is evaluated by the high-fidelity model and stored in the model data repository. As soon as a user request for evaluating a set of design parameters is received the FMS performs its checks and routes the request to eh appropriate model, to be executed.

4.4.4. Rom Construction and Parameter Updates

When there is sufficient data from high-fidelity models, the ROM construction service begins processing data on the timed-execution basis as explained in section 3. This service constructs a ROM based on the POD technique (Suram and Bryden, 2015) and once this is

completed, the resulting ROM parameters are stored in the ROM parameter store. These parameters are associated with a unique ID that corresponds to the model for easy and efficient retrieval by the ROM evaluation service.

Once there are ROM parameters associated with this model are stored in the meta-data store, user requests can be evaluated based on these parameters. The ROM evaluation service retrieves the parameters for the model bases on its unique ID and proceeds to perform a POD computation. On completion the temperature and velocity values are returned, in addition to the energy captured by the ROM.

4.5. Discussion and Results

The remainder of this section discusses the results from using this framework to solve this problem. Figure 4.9 shows the time-line of actions performed by a user and the results of the actions taken by the FMS and the services associated with the hybrid model. Based on the user inputs and the substitution rules, the hybrid model results vary as additional models were added to the repository.

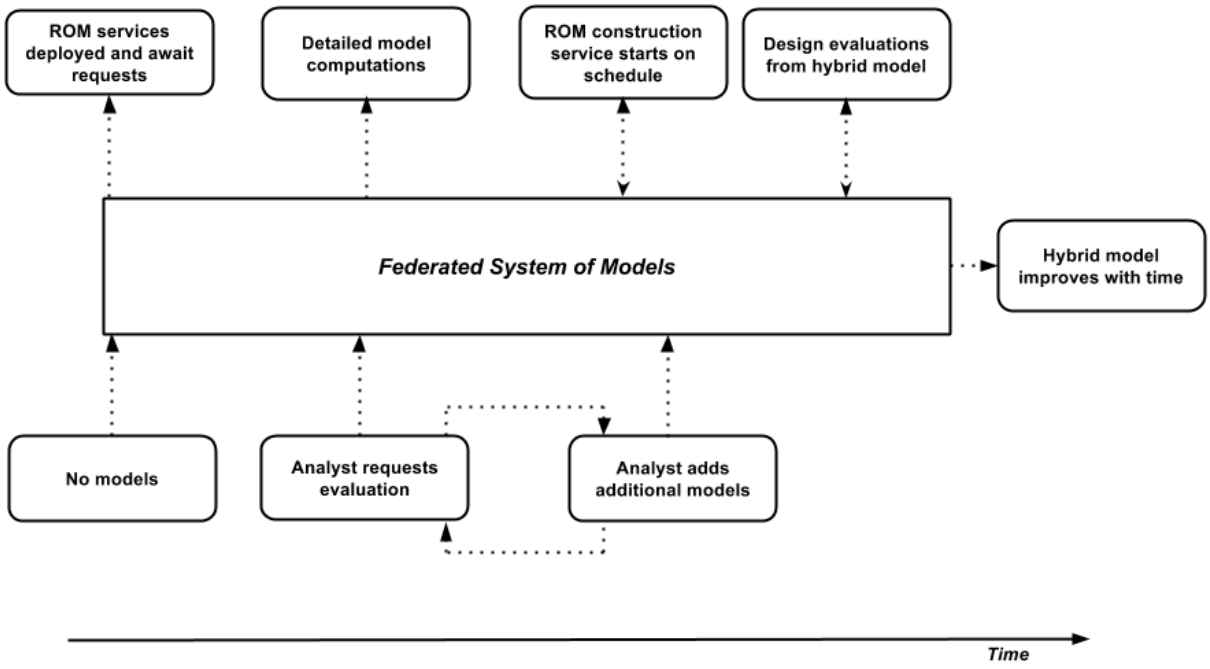

Figure 4.9. Timeline of computations and user interactions with the federated system of models.

Initially, only the detailed model exists after its deployment. Concurrently the ROM construction and evaluation services wait for data and evaluation requests. At this point in time the hybrid model consists only of the detailed model service, since no data has yet been generated to create a ROM. As analysts request the FMS for design evaluations, they get directed to the detailed model. Periodically the ROM construction service checks the repository if there is sufficient data to create a ROM. When there is sufficient data, it proceeds with ROM construction and on completion updates the ROM meta-data repository with the ROM parameters.

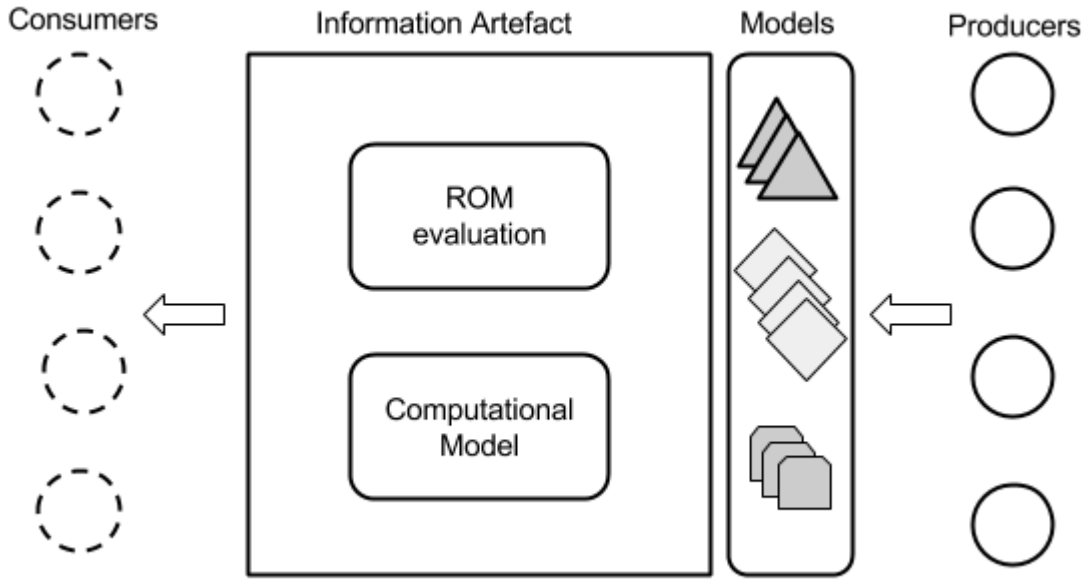

Figure 4.10. Interactions between producers and consumers with information artefacts.

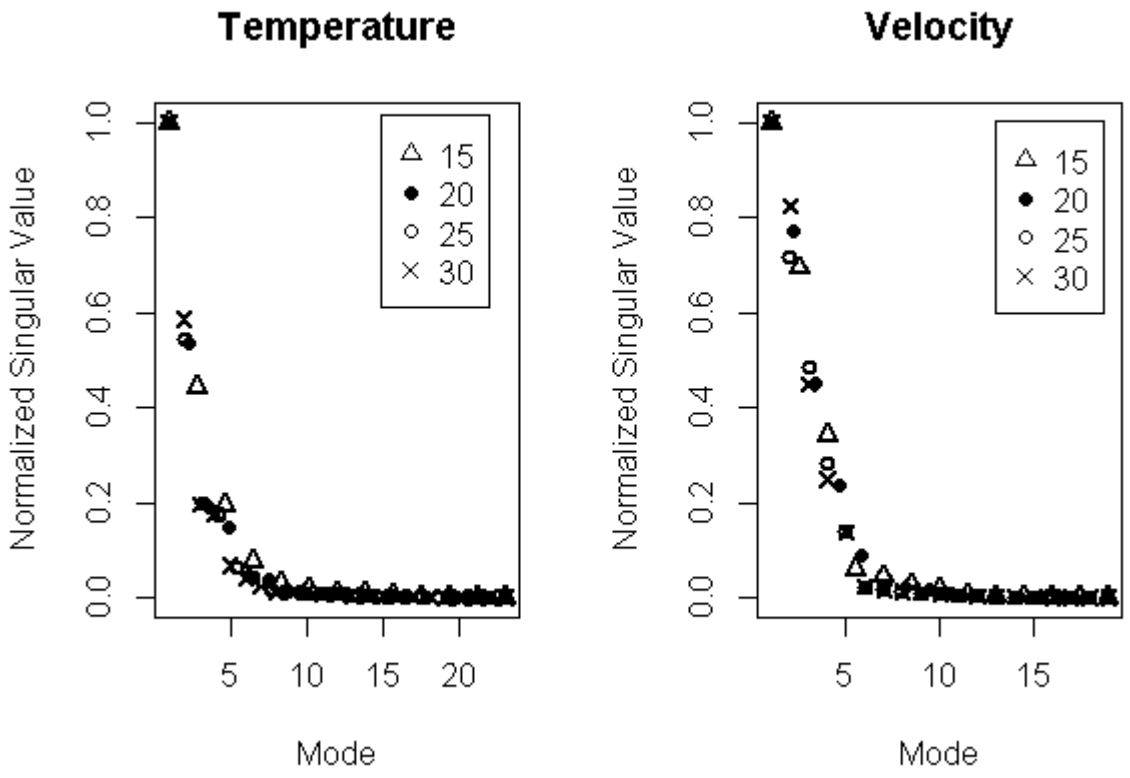

Figure 4.11. Evolution of singular value spectrum with number of detailed models added.

Once the ROM parameters are updated, the hybrid model now consists of both the detailed model as well as the ROM. Hence, the ROM evaluation service can also participate in design parameter evaluations by engineers and analysts. Figure 4.10 shows the information artefact as a single information entity that encompasses the ROM and computational models. As seen, multiple models that are stored in the model repository can be accessed by the IA. As the producers add more models, ROMs get created and the consumers have access to the computations as well as the resulting analysis.

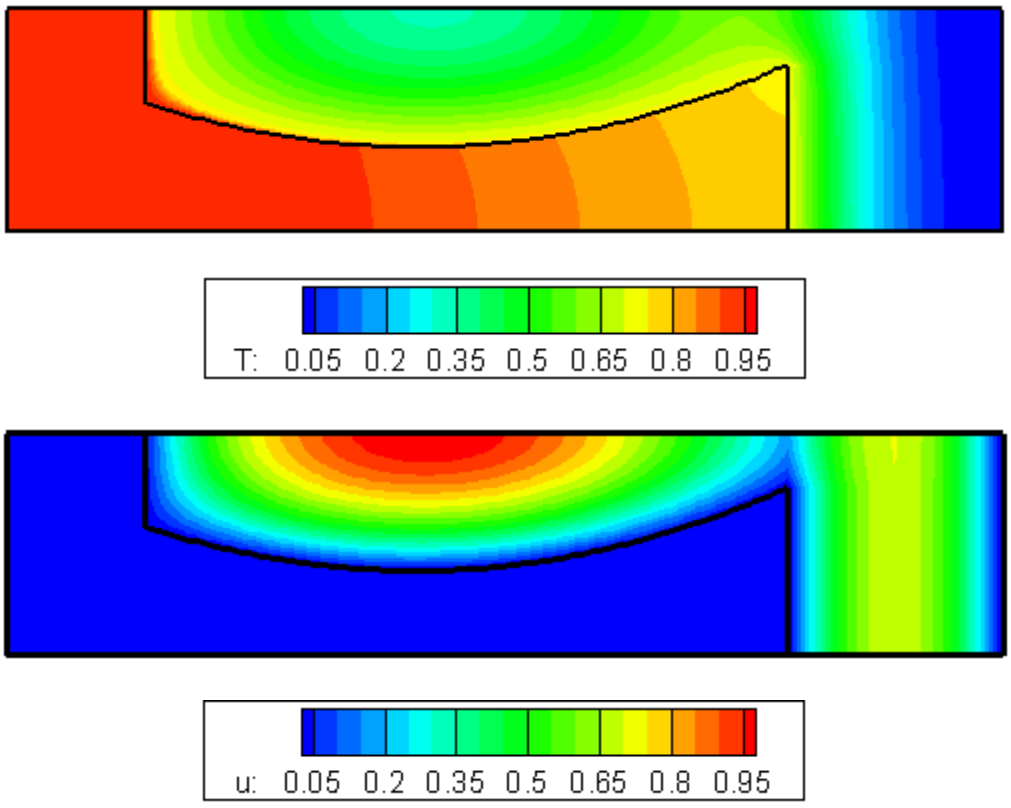

Figure 4.12. Evaluations of temperature and velocity.

4.6. Conclusions and Future Work

In this article, the concept of information artefacts was introduced which are a layer of abstraction over computational models, ROMs, closed-form solutions etc., that are web-enabled. Thus, information artefacts when queried for information about a model return information to the user based on the components that constitute it. In this article, a ROM and a high-fidelity model were considered within an information artefact. The example considered, demonstrates the ability of the artefact to substitute between the constituent models to provide an answer to the user.

In the current work, the process of validating compatibility of models has not been automated. This is an important aspect and can be improved by using a programmatic approach and automating it. One important aspect of automated validation is to ensure that units used in the inputs and emitted in the outputs are the same across compatible models.

References

Pahl G., Beitz W., Schulz H.-J., Jarecki U. Wallace, Ken, Blessing, Lucienne T.M., 2007. (Eds.), Engineering Design: A Systematic Approach, 3rd Edition, Springer Verlag.

Ertas, A. and Jones, J. C., 1996. The Engineering Design Process 2nd ed, Wiley.

Bryden, K. M., 2014. A Proposed Approach to the Development of Federated Model Sets, Proceedings of the 7th International Congress on Environmental Modelling and Software.

Suram, S. and Bryden, K. M., In Review, 2016. A distributed systems approach to engineering modeling.

Suram S. and Bryden K. M., 2015. Integrating a reduced-order model server into the engineering design process, Advances in Engineering Software, vol. 90, pp. 169-182.

Fox A., 2011. Cloud computing – What's in It for Me as a Scientist, Vol 331, Science, pp. 406-407.

Erl, Thomas. Service-oriented architecture: concepts, technology, and design. Prentice Hall, 2005.

Thönes J., 2015. Microservices, IEEE Software, vol. 32, no. 1, pp. 116-116.

Babel, 2016, http://csdms.colorado.edu/wiki/Help:Tools_CSDMS_Handbook (Retrieved 25 March 2015).

Peckham, Scott D., Eric, W. H. Hutton, and Norris, Boyana, 2013. "A component-based approach to integrated modeling in the geosciences: The design of CSDMS." Computers & Geosciences 53: 3-12.

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

In this chapter a summary of the research work completed is presented along with a discussion of the potential impacts of the research. There are also several areas for improvement, further investigation, and analysis which are discussed in the future work section.

5.1. Conclusions

The work presented in this dissertation constitutes three major themes:

- Reduced Order Modeling Server: The reuse of high-fidelity computational data for constructing ROMs to enhance communication of engineering data and models between the producers and consumers of the models.
- Distributed Systems based Engineering Modeling: A proposal to decompose monolithic models into smaller reusable components that are web-enabled. As part of this, this dissertation implements and demonstrates a federation management system which helps link models and orchestrate the flow of information between them.
- Compound or Hybrid Models: The concept of a compound or hybrid model is introduced in the context of an engineering model composed of information artefacts. A compound or hybrid model as discussed is a standalone model composed of two or more models, each with differing approaches to delivering the same information. Depending on the request received and the availability of information, any one of the information sources (models) may provide a response with the requested information. In the example developed here, a high-fidelity computational model and a ROM are constituents of an information artefact (compound model) that responds to queries about the engineering model.

Each of the above themes are summarized in the remainder of this section.

5.1.1. Reduced Order Modeling Server

While ROMs have been used to reduce the time-to-solution of complex and time-consuming problems, a novel use for ROMs in the form of a ROM server was introduced in this research. The driving requirement for this is the difficulty of sharing engineering data. Specifically, in distributed engineering teams where there are multiple producers and consumers of engineering models and data, sharing the models and data is a challenging problem that compounds as the team size increases.

To address this issue, a ROM server has been developed that consumes data from high-fidelity computational models to construct POD based ROMs. A server based approach was taken so that clients have the ability to connect to the server and request engineering design evaluations, over a network. This approach decouples the tight interactions between designers (consumers of models) and analysts (producers of models). In an iterative engineering process this decoupling decreases the design time considerably because the designers and analysts do not have to synchronize their data explicitly on a periodic basis. In this research, an analysis of the cost of synchronization between multiple producers and consumers has been presented. As part of the analysis, comparisons have been made based on the absence and utilization of a ROM server. It was found that the ROM server decreases the cost of synchronization between producers and consumers of engineering models and data. As analysts add engineering models to the ROM server, the POD-based ROM parameters are periodically recomputed and stored. Based on these parameters and knowledge of design parameters, the implemented ROM server is capable of evaluating designs that are within the design parameter range. Thus, once the ROM parameters are known designs can be evaluated quickly without invoking time-consuming high-fidelity models. As the design space changes over time, data can be updated, added or removed

as needed, by the design team. The ROM server thus provides a consistent view of the engineering models, data, and analysis as well as performing designs evaluations in an efficient manner. Additionally, the ROM server also track the "energy" of the POD approximation and can inform users if more data needs to be added to the design space.

A heat-exchanger fin shape design problem was demonstrated using the implemented ROM server. Starting with an initial set of designs a ROM was constructed using the ROM server. Multiple producers and consumers of these models and data were simulated with varying conditions of availability of data and models. These interactions using the ROM server were then evaluated during the course of the design life-cycle to conclude that the ROM server enabled seamless interactions between various producers and consumers of engineering models and data.

5.1.2. Distributed Systems Based Engineering Modeling

Traditionally engineering codes are built using libraries of numerical codes and other integration codes. This approach requires that the model developer be proficient with the corresponding computer language as well as have the ability to understand the inner workings of the libraries as well as the codes being integrated. In this research, an approach based on distributed, statelsss microservices has been proposed, implemented, and evaluated with an example engineering problem in which a larger more complex design code has been divided into a set of smaller models. Each of these models was then implemented as an independent, stateless, and web-enabled microservice. Each of these microservices publish their inputs and outputs, and systems models can be built by linking one or more of these microservices together. Once linked, the systems model can also be published as an independent service. The primary advantage of publishing engineering models as web-enabled services is that the interface to the models is language agnostic and available over the Internet. Thus, a model developer does not need to

know details about a library or programming language to invoke the model, but only an understanding of the input format and the corresponding outputs form the service. Additionally, the model microservices are stateless, implying that state information is not stored after the model computations have been completed. This is an important consideration in this research because statelessness of the models allows their easy reuse across different systems of models and their scalability. The set of component models are then called a federated set of models. In order to orchestrate information flow between these models within a systems model, a federation management system has been developed. Each model registers with the FMS, after which the FMS can send requests for computations to that model. When multiple models need to be invoked in succession, a message that consists of this list can be sent to the FMS, which routes and invokes these models in the requested order. All state information that needs to be passed between models is encapsulated in the message, either directly or as a reference to a location that can be accessed by all services in the federation, say, a distributed file system for large amounts of data. To enable model developers to easily develop models using this architecture, a Model software development kit (SDK) was developed as a part of this research. Using the Model SDK, interacting with the FMS becomes easier as model developers need to implement three function calls (`GetMessage(), ProcessMessage() and SendMessage()`) to encapsulate all the interactions with the FMS and other models registered within that federation of models.

As noted previously a previously developed monolithic numerical model for the preliminary design of cookstoves was split into five stateless models and two stateless functions and incorporated into the developed architecture using the SDK. The primary objective of solving this design problem was to demonstrate the applicability of this research. The results from the monolithic model and the federated set of models were compared for accuracy and

computation time. It was found that both models resulted in the same efficiency of the cookstove for a given set of design parameters. The computational time was, however, higher in the distributed systems model due to network latency introduced by the interactions with the FMS. The effect of network latency can be considerable when the component models have small execution times. However, for component models that have larger computational times the latency effect can be small compared to the computational time of the systems model. The distributed system of models enables model developers to reuse and efficiently build systems models.

5.1.3. Compound or Hybrid Models

An information artefact is a provider of information to a system model i.e., they can be computational models, closed-form solutions, data, design parameters, optimization algorithms, etc. In this context, all these pieces of information are recognized as information artefacts (IAs) that can be queried with an input and a response of returned from them. In this research, a high-fidelity model and its corresponding ROM were combined to form a compound model which then becomes an IA. This IA was then incorporated into the previously developed FMS.

An IA for a heat-exchanger fin was created using the ROM and the high-fidelity computational model. The IA returns the temperature and velocity fields, when queried by the design parameters for the heat-exchanger fin. Under certain conditions the IA invokes the ROM and the more time consuming high-fidelity model is invoked when these conditions are not satisfied. In this case, if the user requests design parameters that are not within the design parameter space of the ROM, the high-fidelity model is invoked. The results of the computation are stored, and once there is sufficient data to construct a ROM, the ROM construction service is invoked and the resulting ROM parameters are stored. Further user evaluations by the IA were

then based on the ROM parameters, by checking if the user query is within the bounds of the ROM. If so, the response is returned to the user based on the results of the ROM evaluation service.

Thus, the IA functions as a hybrid model that encapsulates the entirety of information, models, etc. about a specific piece of information or model within a larger systems model and can evaluate each of them based on user needs. For instance, if a user explicitly requests a high-fidelity model evaluation, the request is evaluated without performing any model substitutions. This hybrid model is particularly useful when long compute times are encountered in a detailed model.

5.2. Future Work

There are several avenues to extend and improve the work presented in this dissertation. These can be categorized under the areas of infrastructure, applications, performance and visualization, each of which is detailed in the following sections.

5.2.1. Infrastructure

This research focused on building a framework for stateless, loosely coupled models to interact with one another through the FMS. The utility of this framework has been demonstrated with example engineering problems. An advantage of stateless models is that they can be scaled-up with relative ease using containerization technologies like Docker. However, for this framework to scale to, say, thousands of constituent models, certain aspects of the FMS need to be improved. Currently models need to be registered manually with the FMS. Incorporating automated registration of model services in the FMS will address this particular scalability issue.

In the current work, for two models to be substituted for one another they need to have the same inputs and outputs. The system builder needs to manually map the inputs and outputs in

order to make the substitution between the models. This process can potentially be automated by incorporating a service that discovers all compatible mappings and suggests the most compatible ones to the system builder.

Yet another very interesting extension to the current research is the development of a domain specific language (DSL). DSLs as the name suggests are languages built for domain experts in a particular field, where the experts can perform their tasks without the need to use a programming language. In this context, there is an opportunity to develop a DSL that can link component models to construct a system model. Since the framework developed in this research requires that the system builder be familiar with some aspects of programming, development of a DSL would be major improvement and can enable wider adoption.

5.2.2. Applications

The framework developed in this research has been utilized for solving examples involving a) computationally expensive, coupled high-fidelity models for the heat exchanger fin and b) coupled first-principles models for the cook-stoves. This research can be extended by solving more complicated engineering problems and coupling them with financial models for price-performance studies on engineering designs. Another example is coupling the cook-stoves model with an economics model to study the economic impact of the cook-stove design at the level of an entire village as proposed by Bryden et al. 2015. Engineering optimization problems would also benefit from the framework developed in this research. For example, the ROM evaluation service can be utilized to compute the objective function when possible. Objective function evaluations requiring the high-fidelity model can be computed asynchronously and the results can be used to update the ROM. This strategy can be especially useful in optimization methods requiring evolutionary algorithms.

5.2.3. Performance

In this dissertation, constituent models have been converted into services that are exposed to the Internet. Examples have been demonstrated of successfully using these services to create complex models based on the constituent models. Although this approach helps engineering models and data to be communicated in an easier manner, it also adds latency due to communications and model interactions over a network via the FMS. For a model built using the "library approach" that has no latency issues, this added latency cost can seem to be very high. However, this depends on the computational time of the model. For models with high computational time, small increase in communication time can be acceptable. On the other hand, for models that have a small computation time, added communication time increases the overall time-to-solution. Although this can be acceptable in many cases, it can also make the solution using this approach unacceptable in some cases. More studies need to be performed to understand, in this context, the relationship between communication and computation times. Also, if data-locality in data centers is considered, it opens up opportunities for interesting research in optimizing the run-times of the model services depending on the computations.

5.2.4. Visualization and Usability

The primary focus of this dissertation is on the development of an infrastructure level distributed system that can be used to link engineering models for efficient use. An important aspect of making this system usable is at the interface of humans and computers. This is an area that has not been addressed in this dissertation. Thus, there are several areas where major improvements to the current work can be made. One such example is user-interface design for "visually building" composite engineering models using the FMS. Furthermore, since the

framework that has been developed is loosely-coupled, the visualization and usability extensions can be made without changes to the framework itself. This approach will be complementary to the DSL based approach discussed previously. Further research in these areas influence the applicability of this research to practical engineering problems in the industry.